Universidade de Aveiro Departamento de Engenharia Mecânica
2019

João Pedro Martins dos Santos

Smart Object Exploration by Robotic Manipulator

Exploração Inteligente de Objetos Por Manipulador Robótico

Universidade de Aveiro Departamento de Engenharia Mecânica
2019

João Pedro Martins dos Santos

Smart Object Exploration by Robotic Manipulator

Exploração Inteligente de Objetos Por Manipulador Robótico

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Miguel Armando Riem de Oliveira, Professor Auxiliar do Departamento de Engenharia Mecânica, e de Rafael Lirio Arrais, Investigador do Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência da Faculdade de Engenharia da Universidade do Porto e de Germano Manuel Correia dos Santos Veiga, Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto.

o júri / the jury

presidente / president

Professor Doutor Jorge Augusto Fernandes Ferreira
Professor Auxiliar da Universidade de Aveiro

Doutor Luís André Freitas da Rocha
Investigador do *Instituto de Engenharia de Sistemas e Computadores do Porto*

Professor Doutor Miguel Armando Riem de Oliveira
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Ao Professor Miguel Riem Oliveira o meu muito obrigado por todo o apoio, incentivo, conversas, disponibilidade e ensinamentos ao longo destes meses. A sua paixão pela programação é contagiante. Um orientador no verdadeiro sentido da palavra.

Ao Engenheiro Rafael Arrais, por toda a valiosa ajuda, experiência e paciência durante todo o desenvolvimento e, especialmente, pelo excelente acolhimento nas duas semanas em que me desloquei ao INESC TEC, o meu sentido obrigado.

Aos meus pais, Cristina e José. Sem eles, não teria sido capaz de alcançar absolutamente nada. Devo-lhes tudo, com a garantia que nunca me cobrarão nada. O maior agradecimento é para vocês, meus pais.

À Vanessa, claro, um especial obrigado, por simplesmente tudo. O suporte nos momentos mais complicados, os inúmeros momentos de carinho, paciência, descontração, conversa ou de um simples "tu consegues" são fundamentais, tanto nesta dissertação como na vida. Sem ti este caminho teria sido, sem dúvida, muito mais custoso. Mereces o mundo.

Aos amigos que, desde o primeiro ano, comigo enfrentaram as adversidades lado a lado. Guedes, Mariana, Miguel, Pina, Teresa e Tiago, este obrigado é para vocês.

Ao Tiago Almeida e ao Tiago Tavares pelas discussões, apoio e confraternização ao longo destes cinco meses. Também ao Eng. Rui Heitor por todo o suporte logístico.

Por fim, uma palavra de agradecimento ao Eng. Rui Cancela e ao Eng. Gonçalo Carpinteiro pela ajuda com a actualização do robot.

keywords Autonomous, Calibration, Environment Representation, Exploration, Next Best View, RGB-D, ROS, Robotic Manipulator, Voxel

abstract The end goal of this dissertation is to develop an autonomous exploration robot that is capable of choosing the Next Best View which reveals the most amount of information about a given volume.
The exploration solution is based on a robotic manipulator, a RGB-D sensor and ROS. The manipulator provides movement while the sensor evaluates the scene in its Field of View. Using an OcTree implementation to reconstruct the environment, the portions of the defined exploration volume where no information has been gathered yet are segmented. This segmentation (or clustering) will help on the pose sampling operation in the sense that all generated poses are plausible. Ray casting is performed, either based on the sensor's resolution or the characteristics of the unknown scene, to assess the pose quality. The pose that is estimated to provide the evaluation of the highest amount of unknown space is the one chosen to be visited next, i.e., the Next Best View. The exploration reaches its end when all the unknown voxels have been evaluated or, those who were not, are not possible to be measured by any reachable pose.
Two case studies are presented to test the performance and adaptability of this work. The developed system is able to explore a given scene which, initially, it has no information about. The solution provided is, not only, adaptable to changes in the environment during the exploration, but also, portable to other manipualtors rather than the one used in the development.

palavras-chave

Autónomo, Calibração, Representação do Meio Ambiente, Exploração, Próxima Melhor Vista, RGB-D, ROS, Manipulador Robótico, Voxel

resumo

O objetivo final desta dissertação é desenvolver um robot de exploração autônomo capaz de escolher a Próxima Melhor Vista que revela a maior quantidade de informações sobre um determinado volume.
A solução de exploração é baseada num manipulador robótico, num sensor RGB-D e em ROS. O manipulador proporciona movimento enquanto o sensor avalia a cena no seu campo de visão. Usando uma implementação OcTree para reconstruir o ambiente, as partes do volume de exploração definido onde nenhuma informação ainda foi recolhida são segmentadas. Esta segmentação (ou agrupamento) ajudará na operação de amostragem de poses no sentido em que todas as poses geradas são plausíveis. Ray casting é realizado, seja com base na resolução do sensor ou nas características da cena desconhecida, para avaliar a qualidade da pose. A pose que é estimado fornecer a avaliação da maior quantidade de espaço desconhecido é a escolhida para ser visitada em seguida, ou seja, a Próxima Melhor Vista. A exploração chega ao fim quando todos os voxels desconhecidos tiverem sido avaliados ou, aqueles que não o foram, não sejam possíveis de serem medidos por qualquer pose alcançável.
Dois casos de estudo são apresentados para testar o desempenho e adaptabilidade deste trabalho. O sistema desenvolvido é capaz de explorar uma determinada cena sobre a qual, inicialmente, não tem informação. A solução apresentada é, não só, adaptável às mudanças no ambiente durante a exploração, mas também, portável para outros manipuladores que não o utilizado no desenvolvimento.

Contents

List of Tables

Intentionally blank page.

List of Figures

Intentionally blank page.

List of Algorithms

Intentionally blank page.

Acronyms

API Application Programming Interface. 84

DoF Degrees of Freedom. 20, 25, 95

FOV Field of View. 12, 16, 66, 71, 80

FPS Frames per Second. 26

INESC TEC Institute for Systems and Computer Engineering, Technology and Science. 84, 87

LAR Laboratório de Automação e Robótica. vi, 30, 36, 43

LSD-SLAM Large Scale Direct SLAM. 21

NBV Next Best View. vii–ix, 3, 5, 11, 14–16, 23, 66, 68, 71, 73, 75, 78, 80, 83, 85, 86, 95–97

NDT Normal Distribution Transform. 19

PCL Point Cloud Library. vi, 30

SLAM Simultaneous Localization and Mapping. 21

SRDF Semantic Robot Description Format. 35

TCP Tool Center Point. 22

URDF Unified Robot Description Format. 32

VHF+ Vector Field Histogram Plus. 18

Intentionally blank page.

Chapter 1

Introduction

Object recognition by robotic systems is a key part of many industry-wide applications and services. Despite increasingly efficient, this recognition is often based on the processing of information collected at a single moment, such as images or point clouds. Thus, most implementations assume that the object recognition process is not dynamic.

To make this process actually dynamic, a system needs to have some kind of exploration abilities, i.e., being able to search and discover information about a given goal.

The increasingly more dynamic and smart implementations will force a paradigm shift in the field of exploration by intelligent systems, since these give a new level of adaptability to new automation tasks that, at this point, are not possible. These implementations can range from a smarter bin-picking, reconstructing the pieces at each iteration for a better adaptation, to a system which collaborates with humans, detecting when something has been placed or removed in its working space, reacting accordingly. For this to happen the new solutions need to be at least as reliable, fast and secure as whats already implemented.

As humans this process is intrinsic to us. The ability to explore a new environment, fast and efficiently, developed in order to increase our survival aptitude. Our eyes evolved to catch the tiniest of details and our brains got smarter in understanding the environment and processing it in order to evaluate the goal, extrapolate the information gathered and choose where to move next.

This raises the question: how to give a piece of hardware, that can not think for itself, the human abilities of knowing what it has and has not already seen and understanding from where it should observe the scene next in order to gather the most new information available?

This challenge leads us to two distinct but interlaced problems: how to see and how to know.

Going back to the human analogy, our brains have almost unlimited storage capabilities to the point that we can close our eyes and see a room or place, that has already been explored, with much details has pleased, as if we were in it. This richness of information is incomprehensible in a modern day-to-day computer. Either we have a very detailed model of the space that is too large to make any use of it, or we have some kind of environment discretization, loosing detail but relieving computational overhead for other tasks.

Another question is that environments are subject to changes. When there is an inconsistency between the real world and the world model, without even notice we detect

it, generate and store a new model with the updated information. This adaptability is also true for new scenes where each new view adds information to a new model being recreated and understood, filling the gaps that are unknown. This process is indispensable in exploration so its fundamental to mimic it. The idea is to fuse a compact representation of the discretized scene to a highly updatable model that also carries information about what it does not know yet.

This introduces the second problem, how to know. Since information about what is unknown is available, the task resumes itself to choosing a view pose (i.e., position plus orientation), evaluate it and decide if it is the best one or not. Once again we deal with the problem of the world being a continuous reality and computers not being so good dealing with that. The solution is again discretization. Using a sampling method solves both these difficulties, the pool of plausible pose choices is now finite and having various of them makes possible to rank them in comparison with the others.

The following section describes the milestones established to accomplish the end goal of this dissertation, creating a autonomous system capable of exploring a given volume or environment.

1.1 Objectives

In this section the proposed objectives for this master dissertation are briefly described and there usefulness is explained with focus on the final integration with the autonomous system.

1.1.1 System ROS Based Drivers

As discussed previously, our ultimate goal is to conduct environment exploration using a robotic system which is composed of a manipulator with a RGB-D camera assembled on the gripper link. Thus, the first objective is to establish communication with all the hardware involved. This includes installing and testing the drivers of the RGB-D cameras and well as the manipulator drivers. Furthermore, in the case of the manipulator, it is not only necessary to collect information from the joint states, but also to be able to properly command the robotic arm. For this we will use the a trajectory planning api called MoveIt. All these functionalities should be embedded into a ROS based ecosystem.

1.1.2 System Calibrations

The autonomous system ability to explore a scene depends on two, more basic, capabilities: sensing the environment and changing its point of view. Each one of them is done with two, independent but coupled, hardware components, the camera and the manipulator itself. Since the goal of this work is to have an autonomous system capable of exploring, these two capabilities must work together.

To do so, the manipulator must know from where the camera is observing the scene and the camera must know where it is in relation to the world coordinate frame – it is assumed to be the manipulator's base link.

The system calibrations takes care of that. Extrinsic calibration tells where the components are in relation to each other and hence, where the camera is in the world. By comparison, intrinsic calibrations are needed for a more accurate acquisition of frames

by the camera, correcting distortions, irregularities and other issues related with the camera's lens.

Once correctly calibrated, the point cloud data, acquired by the depth sensor, will be correctly integrated in the reconstruction model and will allow the system to improve it with multiple views.

1.1.3 Environment Updatable Representation

With the motion of the robot, a new view of the object is available. This new view feeds the system with more information. Maximizing each new view knowledge is imperative to the end goal of this dissertation.

In each position, and even while moving, the camera will gather a new point cloud from its field of view. Because the calibration is already been made, this data will update the correct voxels in the OctoMap [Hornung *et al.* 2013] creating a better model of the environment (and the objects in it).

Still, there is the problem of choosing which will be the next viewing pose.

1.1.4 Scene Exploration

Since multiple views are requested, there must be a way to choose the Next Best View (NBV). This can be done in several ways, as discussed in section 2.2. Most of the techniques consist in optimizing an equation (usually denominated of utility function) that is based on the potential information gain from a position and the cost to reach it.

The process will be finished when the amount of information given by the NBV is below a given threshold. This implies that none of the voxels are in an unknown state, or that the ones who are cannot be visualized within the reach of the manipulator.

1.2 Reading Guide

This document is divided in five main chapters: 1: Introduction, 2: State of the Art, 3: Proposed Approach, 4: Results and Discussions and 5: Conclusions and Future Work.

In Introduction, the general problem is described leading to the State of the Art, where several approaches are presented emphasizing their strengths and shortcomings.

Next comes the Proposed Approach chapter, where the hardware and software necessary to accomplish the objectives is described. Then, the methodology, roadblocks and solutions developed are described. This includes the manipulator's configuration to accept external commands, the mounting and calibration of the camera and the exploration algorithms.

In Results and Discussions, the author introduces two case studies where the developed work is tested, following a comparison between the autonomous system and several humans abilities to explore a given scene. Still in this chapter, plausible implementations and further improvements are discussed. Lastly in chapter Conclusions and Future Work, conclusions are outlined.

This document also includes three appendix: A containing the technical drawing of the camera mount design to couple the camera to the manipulator, B with the transformations tree of the robot and C has the connections between the ROS nodes and topics.

Intentionally blank page.

Chapter 2

State of the Art

As discussed in chapter 1 the focus of this dissertation is to solve the Next Best View (NBV) problem on a robotic manipulator. In this chapter it is studied what has already been done in these fields that can be helpful to solve the problems associated with an autonomous exploration system. These problems range from the calibration procedures that are needed for the full integration between the manipulator and the camera, to the pose sampling methods and, also, the metric used to choose which of those poses should be visited.

Next, it will be presented several approaches used for volumetric representation of the environment.

2.1 Environment Representation

The first step in the robot's decision making process is to evaluate the world around it, which requires the existence of a model of the surroundings. This information needs to be stored in some way, one in which the intelligence of the manipulator can work and that can be updated when a new frame is captured with any change in the scene or just visualizing it from another perspective.

As stated by [Hornung *et al.* 2013], point clouds, elevation maps, multi-level surface maps and volumetric representations are some of the most common forms to do so, although 3D triangle mesh grids are also feasible, as demonstrated by the [Kriegel 2015].

Regardless of the method that is used, the input is always a point cloud, so it becomes natural to study how this data can be used for environment representation, starting with the approaches that, from a conceptual point of view, are simpler, to those that evolve on them, becoming more complex.

2.1.1 Elevation Maps

Point clouds are, in a fundamental way, a set of points in 3D space. As discussed earlier, they are the most common form of 3D data acquisition. Although they can be easily acquired, with sensors that are accessible having decent performance, this raw data can, very fast, become hard to process.

To solve this drawback various methods exist. What they try to accomplish is to merge some of the points in the cloud, losing as less information about the space as possible. One approach is to create a 2D grid representation of the surroundings, where

each cell also contains a height value. This form of 3D representation is referenced as elevation maps. Elevation maps can be classified as 2.5D models because they have the 2D position information of each cell, which in turn contain an corresponding height value.

This height value is obtainable by various methods. One of the simplest ones is averaging the height coordinate of all points of the cloud falling inside a given cell, which result in Fig. 2.1. The biggest advantage is the filtering of outlier measurements (this is, data points of the point cloud that are noticeable different of those that also belong to that cell). A disadvantage is its inability to capture overhanging structures, according to [Douillard *et al.* 2010].

Another method is to compute the difference between maximum and minimum height measurements within a cell and consider it occupied if exceeds a predefined threshold. This approach does not make an approximation but is more sensible to noise in the measures.

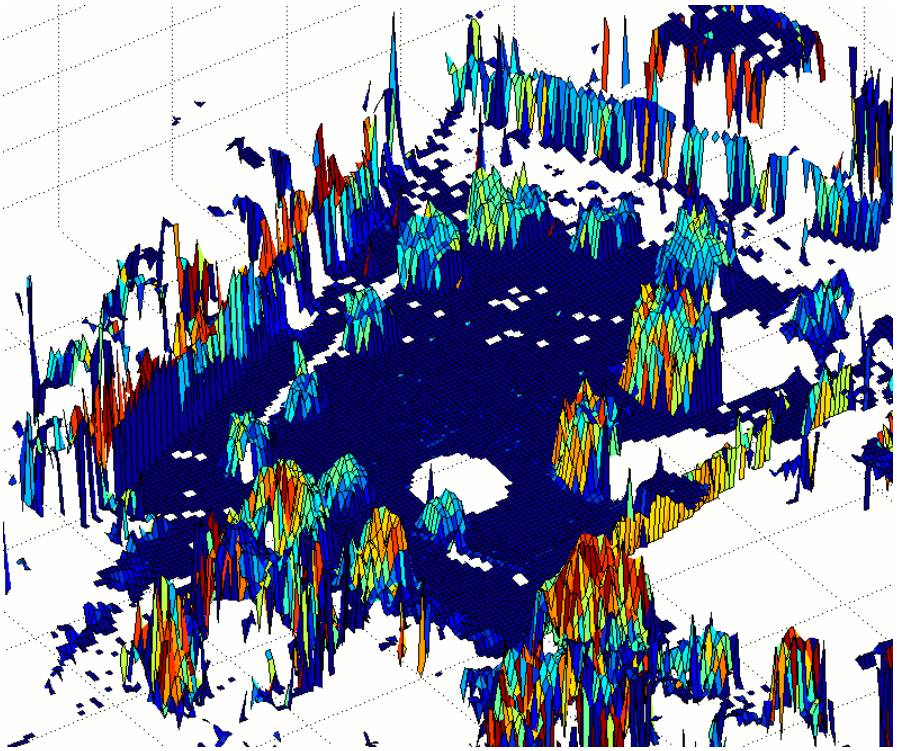
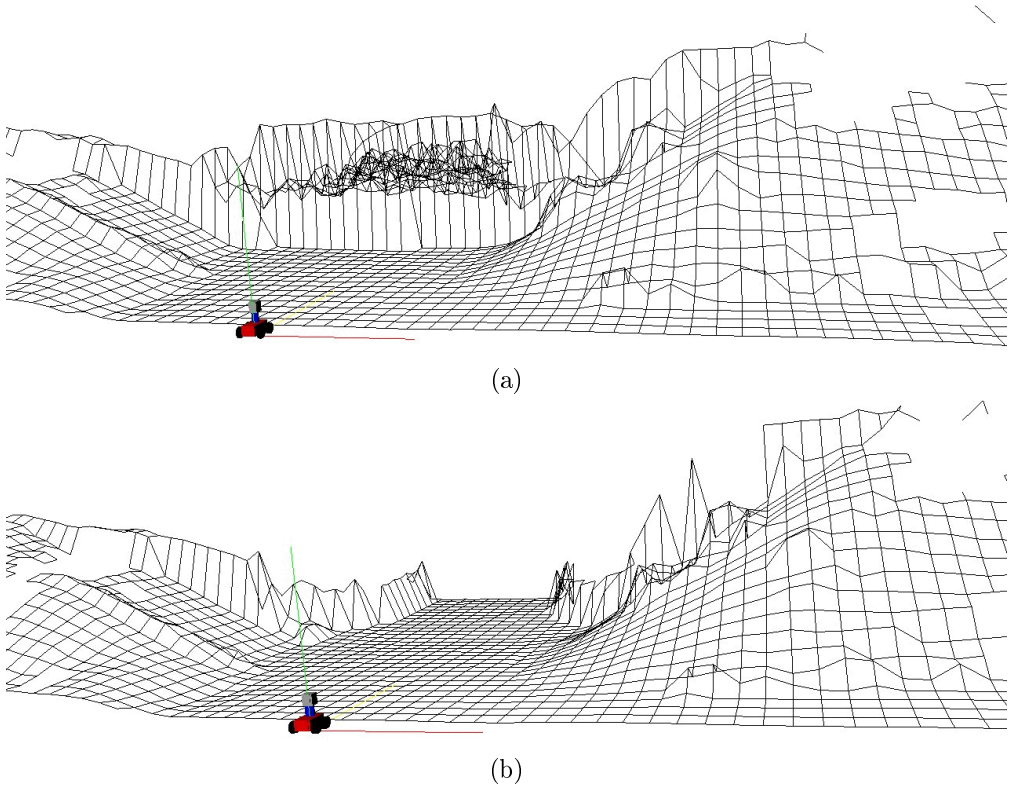


Figure 2.1: Elevation map generated using the mean method. Red indicates a high standard deviation in the measured heights [Douillard *et al.* 2010].

To overcome this inability to deal with overhanging structures, like bridges, [Pfaff and Burgard 2006] developed the extended elevation maps. The indicated approach is able to find gaps by analyzing the variance of the height of all measurements of a given cell. If this variance exceeds a defined threshold, the algorithm checks if this set of points (that fall inside the cell) contain a gap that is greater than the robot they used for testing. If so, the area under the overhanging structure can now be properly represented by ignoring the points above the lowest surface. The difference between a standard and an extended elevation map is presented in 2.2.



(a)

(b)

Figure 2.2: Improvements on dealing with overhanging structures from a standard elevation map (a) to a extended one (b) [Pfaff and Burgard 2006].

Although the extended elevation maps bring improvements to this type of representation, its applicability is limited to works requesting only terrain like representation. This is a major issue in the scope of this dissertation, since a truly 3D representation is needed. For example, if the environment that we want to explore only contains a table, using standard elevation maps would give a fully occupied volume below the table to (similar to Fig. 2.2a). On other hand, an extended elevation map would not properly represent the table top (similar to Fig. 2.2b), not treating it as occupied.

The approaches that are discussed in the following sections use the same idea of dividing the space with a grid but propagate it to 3D.

2.1.2 Voxel Grids

Similar to how, in the elevation maps, was used a grid to agglomerate the data points of the sensed environments, this grid can be scaled to 3D, creating an voxel grid.

A voxel grid (see Fig. 2.3) is generated when the scene is subdivided in several, recursively smaller, cubic volumes, usually called voxels (hence the name voxel grid). This subdivision translates the continuous nature of the world to a discrete interpretation.

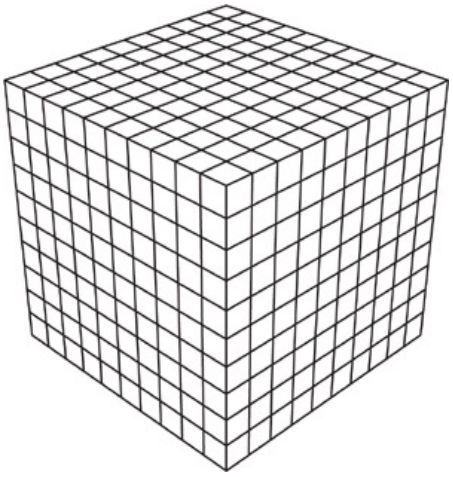


Figure 2.3: A voxel grid. Each cube is a voxel[1].

To represent the scene, each voxel can have one of three different states: free, occupied and unknown. In a newly generated voxel grid, all cells (i.e., voxels) start as unknown since no information as been gathered yet. When a point cloud is received, ray casting from the camera pose to the data points is performed. Ray casting consists in connecting the camera position and a point of the cloud with a straight line. As in Fig. 2.4, all voxels that are passed though the ray do not contain an obstacle that would make the reflection of the ray (i.e., the acquisition of a point) ans so, are identified as free. If, in contrast, a point is sensed inside a voxel, it is considered to be occupied. This is the core of how states are defined but, in practice, for a voxel to be considered occupied it must contain more than a given number of points. The opposite needs to be true for a cell to be free.

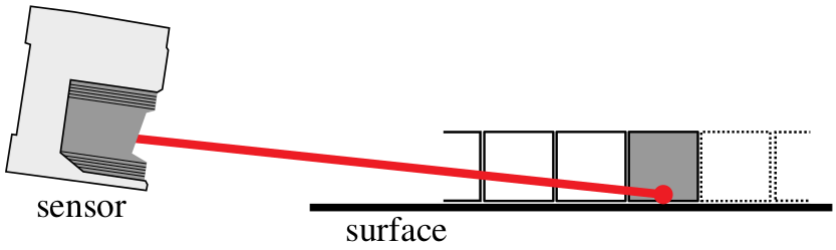


Figure 2.4: Example of a ray cast. Free cells are visualized white, occupied as grey and unknown cells have their border dashed. Adapted from [Hornung *et al.* 2013].

By expanding the concept of grid to a volumetric representation, voxel grids are able to, as accurate as its resolution (the size of a voxel side), reconstruct overhanging structures and virtually any object. Yet they are not a feasible solution as they lack to fundamental characteristics needed, updatability and efficiency. In voxel grids, once a voxel state in defined it can not be changed. This prevents the addition of new information (for example, by viewing by a different pose) to the reconstruction. Furthermore,

[1] From `https://www.physicsforums.com/threads/space-time-distortion-grid-representations.486222/`, accessed on June 4, 2019.

the resolution is fixed, compromising efficiency, especially in cases that need a large voxel grid with fine resolution.

Even with this drawbacks, voxel grids lay the foundations for OcTrees [Maegher 1980], that deal with multiple resolutions within the same volumetric representation.

2.1.3 OcTree

OcTrees builds upon the voxel grids implementation. OcTrees are a data structure that divides the pretended portion of the world into voxels, but allows those voxels to have different sizes.

In an OcTree, each voxel can be divided in eight smaller ones, until the minimum set resolution is reached (see Fig. 2.5). It is this subdividing property of the voxels that establishes a hierarchy between them.

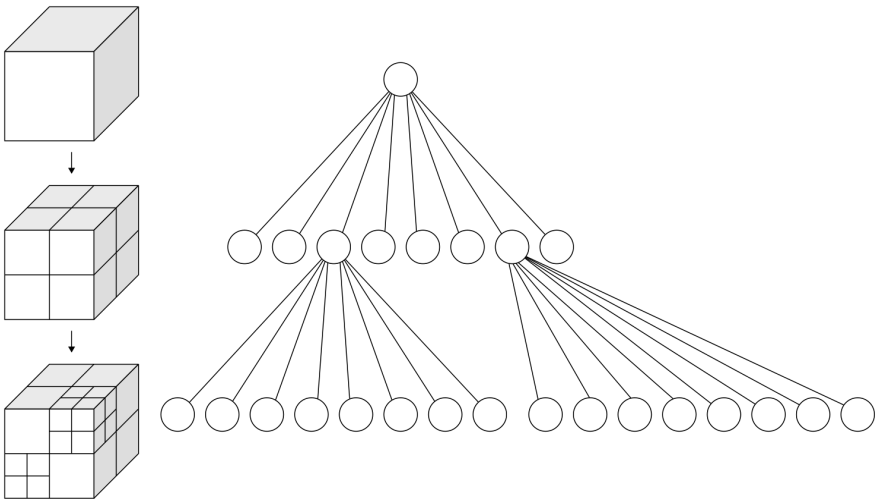


Figure 2.5: Octree data structure. Each node can be divided in eight voxels. If all the leaf nodes are free or occupied, pruning ocurs and those leaf cease to exist[2].

This resolution multiplicity means that some portions of the space that have equal state (this is, are free or occupied) can be agglomerated in one larger voxel (see Fig. 2.6). This is related with pruning. Pruning occurs when all eight leafs (the nodes that have no children) of a node are of the same type, so they can be cutted off, requesting lesser memory and resources, which improves efficiency.

[2] `https://en.wikipedia.org/wiki/Octree` accessed on February 25, 2019.

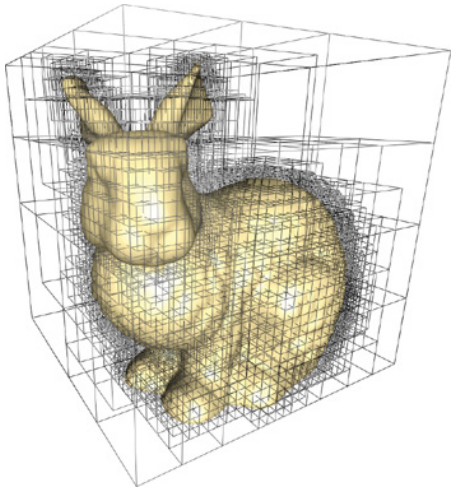


Figure 2.6: OcTree surrounding a 3D bunny model. This shows areas where large voxels exist, namely in empty space, and other areas with smaller voxels due to the higher detail needs[3].

Although the efficiency issue of voxel grids is solved, OcTree still lacks updatability. Updatability creates the difficulty of needing a criteria that leads to a voxel changing its state. In the following section is discussed how probabilities are useful to give this criteria.

Probabilistic Voxel Space

As introduced in section 2.1.2, each voxel is able to store a value that defines if it has been measured to be free or occupied. In the case where that measurement was not taken, the voxel encounters itself in an unknown state. In a probabilistic voxel space, his value ends up being the probability of a voxel being occupied. Since we are dealing with probabilities, a new measure of the environment can be integrated to update those probabilities. With a changing occupancy value comes the possibility of a voxel changing its state. There are several ways in which the integration of a new measure can occur.

OctoMap [Hornung *et al.* 2013] manages this process using the log-odds of a voxel being occupied (equations 2.1 and 2.2), instead of the direct probability value. This causes the substitution of multiplications by sums, achieving a more efficient algorithm.

$$\mathrm{L}(n|z_{1:t}) = \max(\min(\mathrm{L}(n|z_{1:t-1}) + \mathrm{L}(n|z_t), l_{max}), l_{min}) \tag{2.1}$$

$$\mathrm{L}(n) = \log\left[\frac{P(n)}{1-P(n)}\right] \tag{2.2}$$

This formulation is bounded by l_{min}, l_{max} and has its foundations on the probability of eq. 2.3

$$P(n|z_{1:t}) = \left[1 + \frac{1-P(n|z_t)}{P(n|z_t)}\frac{1-P(n|z_{1:t-1})}{P(n|z_{1:t-1})}\frac{P(n)}{1-P(n)}\right]^{-1}, \tag{2.3}$$

[3]From `https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter37.html`, accessed on February 5, 2019.

where z_t is the current measurement, $P(n)$ is a prior probability, $P(n|z_{1:t-1})$ is the previous estimate and $P(n|z_t)$ is the occupation probability of voxel n given the measurement z_t.

In the works of [Kriegel 2015], the OctoMap voxel space update procedure is improved in a way that now considers reflections. Using the MURIEL method (eq. 2.4), the probability of a given voxel being occupied is

$$p = \log(L_{surf}) + \log(L_{free}(1 - P(spec)) + P(spec)), \tag{2.4}$$

where L_{surf} and L_{free} represent, respectively, the occupied and free likelihoods, while $P(spec)$ is referent to measures made in specular surfaces. In addition to this change, [Kriegel 2015] also uses Dynamic Multiple-Octree Discretionary Data Space. This structure allows the use of multiple octrees for discretizing the space.

OctoMap delivers an efficient data structure based on OcTrees that enables multi-resolution mapping. Using probabilistic occupancy estimation, this approach is able to represent volumetric models that includes free and unknown areas, which are not strictly set, meaning that, if enough new frames indicate it, a voxel can alternate its state to accommodate a change in the scenario. As proven by [Kriegel 2015], OctoMap still has room for improvements like being able to consider reflections and to be based in the actual sensor model. Yet it is the most efficient and reliable 3D reconstruction solution currently available.

2.2 Exploration Methodologies

With a found way to accurately represent the environment and its changes, the next logical step is too feed the map with measurements. This measurements could be taken by putting the object/scenario on a rotating platform [Brito Junior *et al.* 2002] or by moving the camera though a set of fixed waypoints. Neither of these solutions is very good in terms of adaptability to the environment, being useful only in a few situations.

An intelligent system needs to adapt to anything the it is presented to it, deciding by itself to where it should move the camera, with the goal of obtaining the maximum possible information about the scene. In some particular cases, part of the scene can not be observed, for example, a box with all six sides closed. The system has to understand when it found a situation like this and give the reconstruction process as completed. Because, in all studied approaches, the process of choosing the NBV and the criteria that leads to the conclusion of the process are so intricately connected, this section presents both proposed methods.

In [Gedicke *et al.* 2016] the goal was to find a sequence of views that lead to the minimum time until the object was found. To achieve this, they determine the ratio between the probability to completely find the target – with a given position – and the transition time that takes to get there, as given by eq. 2.5. In this equation, s_i is the observation being evaluated, L is the sequence of observations done, $p(s_i|L)$ is the probability of finding a target with the observation s_i after the sequence L and $t(L_n, s_i)$ is the time that takes the robot to go from the previous position to the point that allows observation s_i.

$$util(s_i|L) = \frac{p(s_i|L)}{t(L_n, s_i)} \tag{2.5}$$

The innovation in this work is that they plan several steps ahead and, after each pose evaluation, the list of poses that are intended to be visited is updated. By using, as the authors describe, "the probability to find a target with observation s_i after having already seen the sequence L" they assume that all possible positions are determined by sampling (see Fig. 2.7), and so the problem is not only to generate them (see Fig. 2.7), but also to choose which ones to visit and in what sequence [Sarmiento *et al.* 2003].

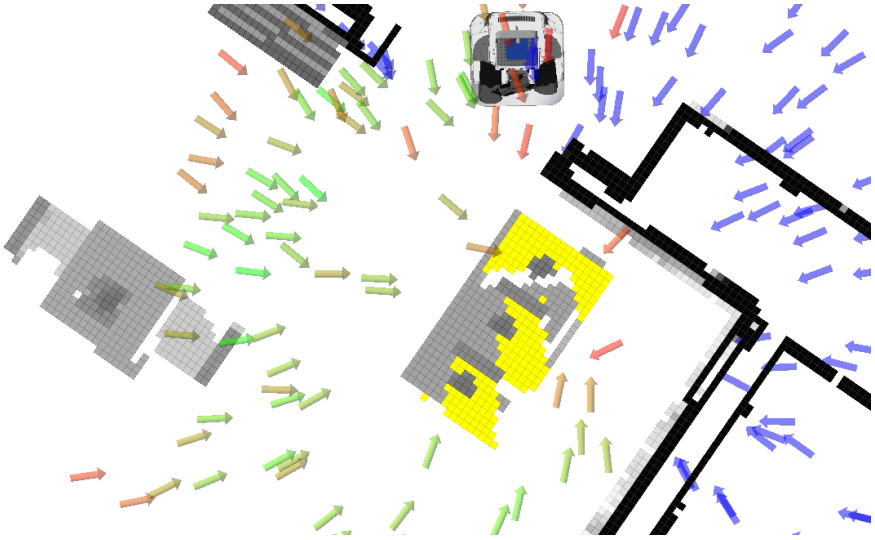


Figure 2.7: Possible view poses (green have high information gain, red have low and blue have none) to observe the region of interest (in yellow are the unknow voxels) [Gedicke *et al.* 2016].

With the utility function 2.5 the planner will favor three situations: i) locations that are closer to the previous position, ii) locations where the probability of finding the object is very high or iii) a combinations of the previous two points.

Since a replanning is done in every pose, a condition to stop the planning procedure must exist. This planning ends when the probability of finding the unknown voxels, by adding a new pose, falls bellow a user-given threshold w, as evaluated by eq. 2.6, in which $p(\hat{V}(S))$ is the notation for the total expected probability to see a target using all views, $p(L)$ represents the total success probability for that plan and w is the termination threshold, i.e. the whole searching is terminated when $p(\hat{V}(S)) < w$.

$$p(L) > 1 - \frac{1 - p(\hat{V}(S))}{1 - w}, \tag{2.6}$$

The described methodology worked very well on [Gedicke *et al.* 2016] experiments. Yet, taking in account that this dissertation will use a robotic manipulator (that compared with the PR2 robot is not able to move as further) the moving time component of eq. 2.5 will not reveal a great significance.

There are other methods that involve the entropy of the space within the view frustum (i.e. the region of space that is within the Field of View (FOV) of the camera, see Fig. 2.8). This entropy is the measurement of how many voxels in different states are possibly detectable by the pose being evaluated. The more different they are, the higher the entropy will be. When [Blodow *et al.* 2011] tackled the problem of semantic

understanding of a kitchen's environment, they used a 2D projection of the fringe voxels on the ground plane. Fringe voxels are considered windows to the unexplored space because they are marked as free but have unknown neighbors.

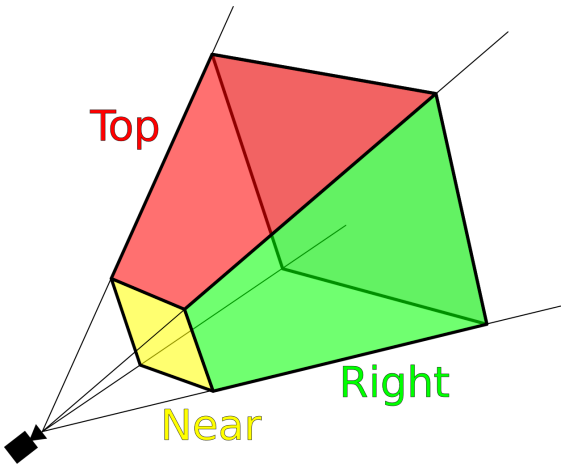


Figure 2.8: The view frustum of a camera[4].

First of all, it is necessary to introduce the concept of a costmap. A costmap is a data structure able to represent places in a grid where it is or is not safe for a robot to move in. Usually, every cell in this costmap has an integer value ranging from 0 to 255, where high values represent high risk of collision, and low values represent low or zero risk. What [Blodow *et al.* 2011] do is to create two separate costmaps for the fringe (C_F) and occupied (C_O) voxels and then combine those values. The combination makes each cell store the minimum value between C_F and C_O. Now, choosing the maximum value from the combined costmap they achieve a pose from which as many fringe and occupied voxels are observed, with little risk of collision. It is necessary to observe several occupied voxels because there is a need to achieve a 50% overlap with this new view and the already created map.

For further pose validation, the entropy H is computed by eq. 2.7,

$$H = -\sum_{i=1}^{2} p_i \log(p_i) \tag{2.7}$$

in which p_1 and p_2 are, respectively, the ratio of the number of occupied and fringe voxels to the sum of both. Multiplying the entropy with its reciprocal reward from the costmap gives a list where the maximum represents the best pose to visit next.

[Blodow *et al.* 2011] do not take in account the moving expenses, which meets the intended for this dissertation, but further adds a new type of voxel (fringe), which makes sense in the large rooms they focused and, furthermore, removes the need to evaluate occlusions, but for object exploration it increases the complexity of system in a direction that will not produce any significant advantage. We need, in each new pose, to evaluate the largest possible amount of unknown voxels, whether they are close to free space or not.

[4] `https://en.wikipedia.org/wiki/Viewing_frustum` accessed on June 4, 2019.

An alternative is to estimate the score of a pose by the volume it could reveal. Similarly to what [Blodow *et al.* 2011] called fringe voxels, [Dornhege and Kleiner 2011] define frontier cells as the ones that are known to be free and are neighboring any unknown cell. Their goal is to remove all unexplored volume, called voids.

For each frontier cell, a set of utility vectors are generated. These vectors contain information about the position of this cell, the direction from the center of the void to the cell, and the expected score value from that cell (i.e., the total volume of the cells that are intercepted with ray tracing, from the center of the frontier cell to the center of the void). For each free cell that is intersected by an utility, ray tracing is performed with its direction, and the utility score stored. Of all these viewpoints,those who are outside the robot's workspace are pruned and the remaining are sorted by there $util(c)$ values for computing valid sensor configurations. By other words, this means – after sorting the viewpoints by their score – sampling some valid camera orientations and perform, again, ray tracing. After a given amount of poses has been computed, the next best sensor configuration is the one which has the highest utility score (i.e. allows the observation of most volume). The recognition process terminates when the last view utility is null.

Another possibility to choose between the sampled poses is not by their total information gain but, instead, ranking them compared to each other. With this goal, [Isler *et al.* 2016] proposes the eq. 2.8 as their utility function.

$$U_v = \frac{G_v}{\sum_V G} - \frac{C_v}{\sum_V C} \tag{2.8}$$

In eq. 2.8 G_v and C_v respectively denote the information gain and robot movement costs of a given view v and $\sum_V G$ and $\sum_V C$ are, also respectively, the cumulative information gains and movement costs of all view candidates.

When a pose gives the maximum relative information gain with the minimum relative cost, the authors consider that they achieved the NBV. When the highest expected information gain of a view falls below a defined threshold, $G_v < g_{thresh}$, the reconstruction is completed.

Until this point, the utility functions are composed by a small number of inputs (one to two), but larger equations are possible.

Trying to get a algorithm that was optimal for motion planning in 3D object reconstruction, [Yervilla-Herrera *et al.* 2018] came up with the goal to maximize eq. 2.9 composed by four distinct inputs: i) position $pos(X_i)$, related to the collision detection, ii) registration $reg(X_i)$, a measurement of overlap between views, iii) surface $sur(X_i)$, which evaluates the new discovered surface, and iv) distance $dist(X_i)$, to penalize the longer paths.

$$g(X_i) = pos(X_i) \cdot reg(X_i) \cdot sur(X_i) \cdot dist(X_i) \tag{2.9}$$

If a position requires a path that enters in collision with the environment, the position portion of the equation is set to zero, eliminating that candidate. The registration part is evaluated as one if the overlap portion of the new view is above a given threshold, or zero otherwise. Next comes the first non-binary value, related to the new surface measured, that gives the amount of unknown voxels sensed with that view, normalized by the amount of the total unknown voxels in the workspace. The final value is inversely proportional to both translation and orientation movements, since the goal is to maximize $g(X_i)$, the smaller the length, the greater this value will be.

To stop the reconstruction process, inequality 2.10 is used, where "m corresponds to the number of sensing operations needed to be certain with confidence $1-\alpha$ to sense $1-\epsilon$ of the portion of the box containing the object", as described by the authors.

$$m \geq \frac{\log(\alpha)}{\log(1-\epsilon)} \tag{2.10}$$

This creates two scenarios, the first in which the percentage of voxels inside the object is known a priori and m is calculated once, or the second one where that value is unknown and so, m is recalculated in every cycle. When $1-\epsilon$ does not change in m operations, the task is considered to be over with certainty $1-\alpha$.

One thing that has been missing in all the described utility functions is a parcel specifically dedicated to the modeling part of the problem. That parcel can be introduced into the equation like in the works of [Kriegel *et al.* 2013] where the exploration and 3D modeling goals are balanced in function of the number of scans taken. The best NBV is the one that gives the most entropy reduction e_p with the most surface quality q_s based on eq. 2.11. The weight w defines which parcel has a higher demand, in function of how many scans have already been done.

$$f_{utility} = (1-w) \cdot e_p + w \cdot (1-q_s) \tag{2.11}$$

The weight w is needed because [Kriegel *et al.* 2013] use a setup with one RGB-D camera and a laser striper. Both combined ally the faster and completer overview of the scene provided from the RGD-D camera with the superior quality from the laser striper. Initially, it is wanted a rough but broad mesh of the object but, latter on, the quality of that mesh begins to matter the most, so a transition value n_q (see eq. 2.12) is chosen to change what is more requested of a view. n_s is the number of scans taken.

$$w = \frac{n_s/n_q}{(n_s/n_q)+1} \tag{2.12}$$

In [Kriegel *et al.* 2013] the stopping criteria was more recognition oriented, so the process was established to end when, at least, 75% of the object was modeled. Yet, also using equations 2.11 and 2.12, the end of the process can be reached when there is already enough mesh coverage and relative point density. If neither is reached, [Kriegel 2015] also implement a maximum number of scans criteria. They also introduce a method for switching from generating candidate poses to rescanning (in their case, rescanning holes in the mesh) if the increase in coverage of one scan compared to its previous is less than 1%.

All the methods already described lay in predefined evaluation functions and human set threshold values. This methods are considered to be a more classical approach to solve a computational task. Currently, the trend to solve such tasks is to use a system inspired on the biological neural networks that constitute brains. Such systems are called neural networks and provide systems with artificial intelligence. These systems can be taught by analyzing examples, like manually labeled data. Analyzing that data, without prior knowledge, and knowing the label they should give as result, these algorithms automatically generate identifiers for that kind of evaluation. With the spreading of neural networks and artificial intelligence, the usage of this new and powerful methods has already been accomplished in some particular situations.

Remembering that each voxel stores as a probability value of being occupied, [Hepp *et al.* 2018] define that the total amount of surface voxels, that have certainty u, in a map M is given by eq. 2.13 where *Surf* is the set of voxels that intersect the object's surface.

$$ObsSurf(M) = \sum_{v \in Surf} (1 - M^u(v)) \tag{2.13}$$

Using utility score function 2.14, which defines the decrease in uncertainty of surface voxels when a new measurement is added, the goal is to predict $s(M, p)$ (that additionally depends of the camera's pose p and the amount of information added with a single measurement η) without accessing the ground truth map.

$$\begin{aligned} s(M,p) &= ObsSurf(M|_p) - ObsSurf(M) \\ &= \sum_{v \in Surf} (1 - \exp(-\eta) M^u(v)) \geq 0 \end{aligned} \tag{2.14}$$

Using a ConvNet architecture, trained with photo-realistic scenes from Epic Game's Unreal game engine, the 3D scene exploration occurs in episodes. An episode starts by choosing a random starting position to mark it as free space. Progressively, at each time step, the set of potential viewpoints is expanded to the neighbors of the current position, all of them are evaluated and the robot moves to the best one. The nine neighbors are defined as six translations in the camera's frame (3 axis $\times$ 2 translation/axis) and three yaw rotations: two of 25^{o} (clockwise and counter-clockwise) plus a 180^{o} one.

The authors claim that their model is able to explore new scenes other than the training data but is dependent on the voxel distribution of the said data. They also do not contemplate moving objects in the scene and are bounded by the resolutions and mapping parameters of the training data.

A more recent attempt, not for scene exploration but to object reconstruction, done by [Vasquez-Gomez and Romero 2019] also implemented a convolutional network. The neural network receives a probabilistic voxel space of dimensions $32 \times 32 \times 32$ and predicts the next best view from a finite possible set. This set creates a sphere of radius 0,4 m around the center of the object, which is also considered the origin of the reference frame.

Seeming to be the first work in this field of its kind, the results achieved were good with the caveat that the data for training and evaluation was divided from the same set, meaning that the extrapolation of the results must be done with care. With this said, the prediction time of this data oriented method is faster then the search methods due to the lack of ray tracing.

The proposed approach for solving the problem planned for this dissertation consists in an adaptation and blend between [Dornhege and Kleiner 2011] pose sampling method, [Isler *et al.* 2016] utility function and [Yervilla-Herrera *et al.* 2018] surface parcel computation. Having a set of points representative of where the camera should look at seams to give a good starting point for the pose sampling procedure, increasing the chance of those pose being plausible, this is, somewhere in their FOV at least one unknown voxel is expected to be evaluated. Comparing and ranking those poses against each other makes it possible to introduce a stopping criteria that directly correlates to the first NBV and so, it is expected that in each exploration iteration (i.e., the movement of the robot to a new viewing pose) the NBV score allways decreases.

The use of neural networks was not considered since their potential in this field has not yet been proved to introduce advantages that overcome the limitations and, maybe even more relevant, that time consuming process of training and evaluating it.

After deciding to where the camera should move, a path to get there must be planed. This path can not enter in collision with the environment, under penalty of changing it, or worse, causing damage on the hardware.

2.3 Path and Motion Planning

Knowing where to move and how to get there are two distinct problems. The first one has already been discussed on the previous section. On this section, the focus is given to the path and motion planing. Path planning refers to the trajectory that the end effector will describe in space, while motion planning is reference to the configuration of each joint, in time, that lead to the end effector actually moving through the planned path.

[Lavalle 1998] proposed the Rapidly-exploring Random Tree (RRT) method which uses a heuristic algorithm to find branches of the current configuration that do not collide with the environment. This method starts in the current joint state and explores the free space by sampling new random states. If this new random state is close enough to some previous configuration belonging to the tree and both can be connected in a collision free trajectory, it is added to the tree.

In the case where there is a final pretended configuration the RRT-Connect [Kuffner and LaValle 2000] process is similar. It creates two trees as shown in 2.9, one initiates from the start configuration and other from the goal configuration, and tries to connect them, planning a motion. This was exactly the goal of [Kriegel 2015] when they opted for the RRT-Connect method.

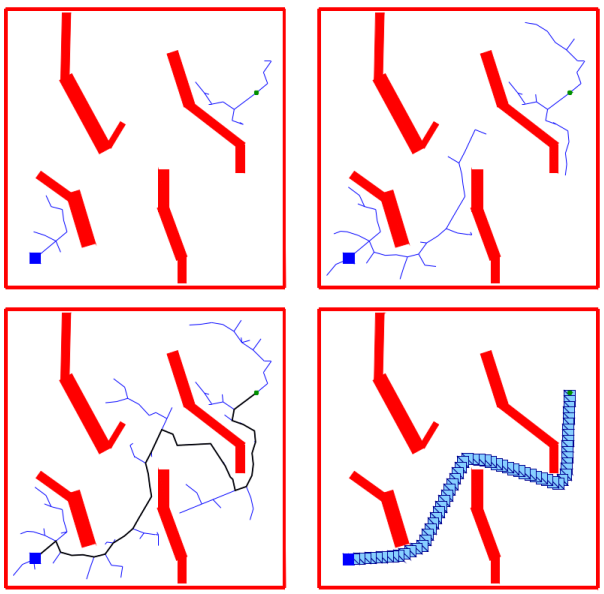


Figure 2.9: Two trees growing into each other by the RRT-Connect algorithm. These trees deviate from obstacles [Kuffner and LaValle 2000].

In more complex tasks, for example when the robot is in motion, either though air or water, collision detection in real time is of the greatest importance. However, the computational cost of this checking to every existing voxel is unbearable.

The algorithm developed by [Vanneste *et al.* 2014] (based on Vector Field Histogram Plus (VHF+), which already is an improvement of the original 2D implementation) tackles this problem by reducing the space used to verify the collisions and optimizes the method of doing so. 3DVFH+ organizes the problem in five different steps.

OctoMap exploring: To limit the range of collision checking, a bounding cube is defined around the vehicle center point

2D primary polar histogram: From this point forward, the algorithm locates a given voxel by two angles around the vehicle center point. For this, the bounding cube is reduced to a bounding sphere, using euclidean distances. The azimuth and elevation angles are calculated. The weight of a voxel is determined based on its euclidean distance (to the vehicle center) and occupancy probability

Physical characteristics: While in motion, if a change in direction is requested, the algorithm must check if the new projected path contains any risk of collision. So, the calculation of both center points of the turning circle is performed to check every voxel that lays within the turning circle

2D binary polar histogram: To further reduce the information contained in the 2D primary polar histogram, a 2D binary polar histogram is generated by simply comparing the primary values to a given threshold that changes according to the elevation angle

Path detection and selection: To select the available paths, a window is moved around the 2D binary histogram. The voxels will only be marked as passable if all of those within the window are registered as zero. The final step is to evaluate the path's weight (the sum of the weights of the passed through voxels) and select the lightest

To calculate a path's weight three parcels are taken into consideration, as described in [Vanneste *et al.* 2014], which are

1. the difference between target angle k_t and candidate direction v;
2. the difference between the rotation of the robot θ and candidate direction v;
3. the difference between the previous selected direction k_{i-1} and candidate direction v,

as denoted in eq. 2.15 where μ_i denotes the impact of each parcel in the final weight.

$$k_i = \mu_1 \cdot \Delta(v, k_t) + \mu_2 \cdot \Delta(v, \frac{\theta}{\alpha}) + \mu_3 \cdot \Delta(v, k_{t-1}) \tag{2.15}$$

This method was simulated on a dataset from [Hornung *et al.* 2013] and one illustrative result is demonstrated in Fig. 2.10.

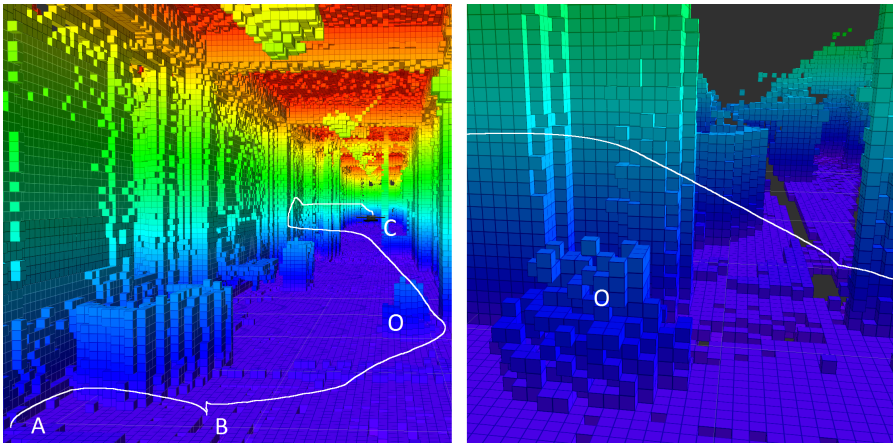


Figure 2.10: Full, simulated, traveled path from point A to point C. The robot was able to deviate from point B and pass over point O (an obligatory waypoint) [Vanneste *et al.* 2014].

Another porting of a 2D algorithm to a 3D environment was accomplished with a Normal Distribution Transform (NDT) [Stoyanov *et al.* 2010], representing the observed points as a set of Gaussian probability distributions. At the start, having the point cloud leads to a creation of an OcTree representation of the world, with a leaf size that is pre-set. For each leaf – and corresponding set of points –, the Gaussian distribution is determined, but some voxels might have points bouncing from different objects, i.e., points that actually do not belong to them. To solve this problem, a heuristic is used on order to obtain better fitting approximations. This reduces the amount of information from all the data relative to the cloud to only nine values per voxel – three for the mean value and six for the symmetric covariance matrix.

Around the moving robot, the voxels within a bounding sphere (with the double radius of the robot) are classified as probably support cells and probably collision cells. This classification is specific to the work in study but can be adapted to other implementations. Then the path is chosen by following the eq. 2.16 gradient, with $\epsilon_i = p_i - \mu$ for each p_i measurement, mean μ and residual variance σ_ϵ^2.

$$fit = \frac{\sum_{i=0}^{N} \epsilon_i^2}{\sigma_\epsilon^2} \tag{2.16}$$

When moving to a new pose, the robot must not collide with what it already knows the existence of. In more confined scenarios this could mean moving inside the object, which required the links to articulate themselves to deviate when the time comes. Because this problem requests a fast planning for every joint, with a high convergence rate, and both the start and end positions (and therefore, resorting to inverse kinematics, configurations too), the RRT-Connect algorithm is the most promising possibility.

2.4 Calibration

Calibration is the process of fine tuning any sensor parameters so the measures it takes are as accurate and precise as possible, compared to the ground truth.

When talking about cameras, there are two types of calibration: intrinsic and extrinsic. Intrinsic calibrations deal with image distortion caused by the lens geometry or even defects. On the other hand, extrinsic parameters give the pose of the camera relative to a reference frame. Both calibrations are static, this is, once they are set it is not intended to change them, unless something happens to the lens or the camera moves relative to reference frame.

Obtaining the intrinsic parameters of a camera is a common task, and so, will not be deeply talked in this section. Although, it is demonstrated how the process was carried out for this dissertation in section 3.6.1. It usually goes by showing to the camera a pattern with known dimensions, in different and not redundant poses. An algorithm then detects some specific characteristics of that pattern (in each shot) to estimate the intrinsic parameters. Figure 2.11 shows the difference between a intrinsic uncalibrated and calibrated image.

Figure 2.11: Image comparission between a distorted (before) and undistorted (after) image. Intrinsic calibration solves the distortion of the before image, giving the after image[5].

Solving the equation systems where the geometric transformation between the moving camera and the reference frame shows up is not simple. In the next section, methodologies developed to retrieve the extrinsic parameters are briefly discussed.

2.4.1 Extrinsic Calibrations

As expressed by [Shiu and Ahmad 1989] the goal of an extrinsic calibration is to find the camera position relative to the robot wrist instead of to other links of the robot, because it is usually mounted to it (last link of the robot) and to allow itself all 6 Degrees of Freedom (DoF). Mounting it in any of the other links will limit the amount of DoF. Furthermore, robot motions are conventionally specified in terms of the position of the last link, therefore it is natural to find the sensor position relative to this link. If there is the need to know the position relative to any other link, it is straightforward to find, using encoder readings and link specifications.

Having the kinematics model of the manipulator, we can directly, by knowing the joint states, compute the position and orientation of the end effector. Knowing the extrinsic calibration parameters, the position of the sensor to this link is a matter of a geometric transformation applied on it.

[5]From `https://www.dronezon.com/aerial-photo-and-video/aerial-photography/remove-barrel-distortion-fisheye-effect-on-aerial-photos/`, accessed on June 5, 2019.

Yet, the other way around is possible, this is, looking up for some features in the retrieved image (the one used to reconstruct the map), track the movement of the sensor. These methods are called Simultaneous Localization and Mapping (SLAM). When using RGB frames for this, there are various ways to detect these features. [Hull 2017] implement Large Scale Direct SLAM (LSD-SLAM) which uses monocular vision to select key-frames and estimating the depth in a subset of pixels by comparing multiple consecutive images in real-time. This sensors can be others than RGB cameras or depth sensors. [Evers and Naylor 2018] successfully used acoustic sensors for SLAM.

Knowing the sensor's pose in the world, the extrinsic parameters allows to compute where the end effector is positioned, relative to the same reference frame.

With this said, in the scope of this dissertation, the method to locate the RGB-D sensor was performed by the kinematics model.

To obtain the referred parameters, according to [Tan *et al.* 2018], there are three possible calibration categories such as robot-world, tool-flange (also known as hand-eye) and kinematic calibrations. Kinematic calibration is often treated as an independent problem of the other two categories and will not be addressed.

2.4.2 Hand-Eye Calibration

Since there is a rigid connection between the end effector and the sensor, it follows that independently of the configuration of the robot, the geometric transformation from one to the other will never change. Thus, we may formulate the problem in terms of homogeneous transformations as in eq. 2.17 in which $\mathbf{T_{6_1}}$ and $\mathbf{T_{6_2}}$ are the geometric transformations from the base of the robot to the end effector for two distinct poses, $\mathbf{OBJ_1}$ and $\mathbf{OBJ_2}$ are the geometric transformations from the camera to a known, fixed, object in the world, for the same two poses and $\mathbf{X}$ is the geometric transformation from the end effector to the camera (see Fig. 2.12). This last transformation is the one that we are searching for.

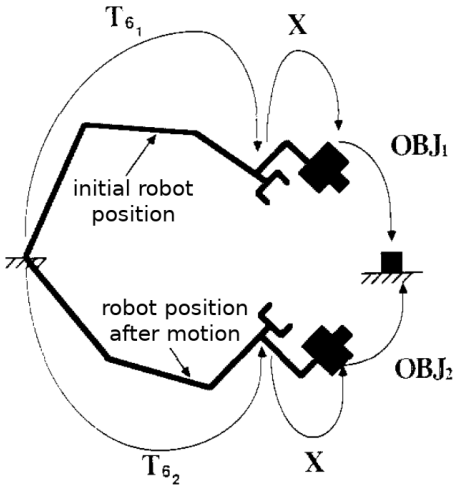


Figure 2.12: Geometric transformations from world to end effector, end effector to sensor and sensor to object, in two different configurations of the robot. Adapted from [Shiu and Ahmad 1989].

$$\begin{aligned} \mathbf{T_{6_1}} \cdot \mathbf{X} \cdot \mathbf{OBJ_1} &= \mathbf{T_{6_2}} \cdot \mathbf{X} \cdot \mathbf{OBJ_2} \\ \Leftrightarrow \mathbf{T_{6_2}^{-1}} \cdot \mathbf{T_{6_1}} \cdot \mathbf{X} &= \mathbf{X} \cdot \mathbf{OBJ_2} \cdot \mathbf{OBJ_1^{-1}} \\ \Leftrightarrow \mathbf{A} \cdot \mathbf{X} &= \mathbf{X} \cdot \mathbf{B} \end{aligned} \tag{2.17}$$

There are many proposed techniques for solving 2.17 that, according to [Condurache and Burlacu 2016], can be characterized as separable, simultaneous or iterative.

Separable solutions are ideal when what is needed is a simple and fast away to get the calibration parameters but accumulated errors in the rotation component can be passed to the position component. To solve this problem, simultaneous solutions appeared but these ones produced variable results on the scaling of the positional component. Iterative methods attempt to solve all these problems but provide no guarantee that their convergent solution was the optimal one. There is no recommended method for every situation, it heavily depends in what the problem consists and what is pretended.

2.4.3 Robot-World and Hand-Eye Simultaneous Calibrations

Also known as "Simultaneous estimation of the hand-eye transformation and the pose of the robot in the world" [Strobl and Hirzinger 2006], this method aims to obtain the parameters of the calibration with four geometric transformations (the used notation corresponds to the one in Fig. 2.13): i) from the world reference frame to the camera's, ${}^{0}\mathbf{T}_{c}$, ii) from the camera's frame to the Tool Center Point (TCP), ${}^{c}\mathbf{T}_{t}$, iii) from the world reference frame to the the robot's base, ${}^{0}\mathbf{T}_{b}$ and iv) from the robot's base to the TCP, ${}^{b}\mathbf{T}_{t}$, giving the relation

$$\begin{aligned} {}^{0}\mathbf{T}_{c} \cdot {}^{c}\mathbf{T}_{t} &= {}^{0}\mathbf{T}_{b} \cdot {}^{b}\mathbf{T}_{t} \\ \mathbf{A} \cdot \mathbf{X} &= \mathbf{Z} \cdot \mathbf{B} \end{aligned} \tag{2.18}$$

In this formulation, $\mathbf{A}$ and $\mathbf{B}$ are known.

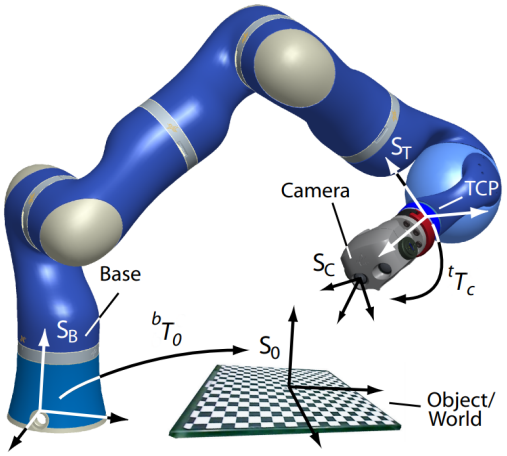


Figure 2.13: Representation of the geometric transformations involved in the robot-world calibration. Adapted from [Strobl and Hirzinger 2006].

Ways to solve both previous approaches can be found on [Park and Martin 1994], [Li *et al.* 2010], [Shah 2013], [Heller *et al.* 2014] or [Tabb and Ahmad Yousef 2017].

2.5 Summary

In this chapter it were discussed many proposed solutions for the needs this dissertation faces in the pursuing of a autonomous exploration system based of a robotic manipulator.

For environment representation, as already stated, we will use the OctoMap framework given its ability to identify space that is yet unknown, while being updatable.

The most amount of development will be on the exploration procedure. This procedure includes poses sampling, NBV choosing and camera movement. Starting from the end, the camera movement is carried out by the manipulator. Motion planners for robotic arms exist, and a develop of a new one is not within the scope of the work, so an already available solution will be used. In contrast, an ideal algorithm for poses sampling was not found, neither for pose evaluation/choosing. This leads us to a development of a new solution that conjugates the strong points of the reviewed works and fit for implementing on a robotic manipulator.

This solution is described in the following part.

Intentionally blank page.

Chapter 3

Proposed Approach

Before starting the development of a new solution, a combination of hardware is required to accomplish the end goal. Some software tools, besides the ones talked in chapter 2, were also available, easing the overall architecture development. A brief description of all is given in this chapter.

3.1 Hardware Implemented

In this dissertation we are dealing with a situation where a given robot needs to volumetric map an arbitrary scene. This requests, in an hardware level, a sensor that is capable of capturing 3D information. For this sensor to be provided with movement, it is attached to a robotic manipulator which will be controlled by the software architecture.

The set of hardware used is as follows.

FANUC M-6iB/6S series is a family of robots with six Degrees of Freedom (DoF) (given by six rotational joints), specially designed for material handling, assembly, picking, packing and dispensing. The M-6iB/6S variant (shown in Fig. 3.1) has a smaller reach of 951 mm compared to the 1373 mm from the normal M6iB (see table 3.3 and Fig. 3.8).

Figure 3.1: The FANUC M-6iB/6S manipulator.

Asus Xtion Pro LIVE (Fig. 3.2) is an RGB-D camera ideal for applications that require motion sensing. It has a 640 x 480 resolution at 30 Frames per Second (FPS) with an ideal distance of use between 0.8 m and 3.5 m allied to a low power consumption. This sensor provides not only a color image but also a point cloud of the scene.

Figure 3.2: The Asus Xtion Pro LIVE RGB-D camera.

Camera mounting A custom support was designed to mount the camera on the manipulator (see appendix A and Fig. 3.3). This support consists of a back aluminum plate that is fixed to a similar plate, already on the end effector, with two bolts. The assembly with the camera in made by a nylon structure that takes advantage from the existing foot mount on the camera to pass through another bolt.

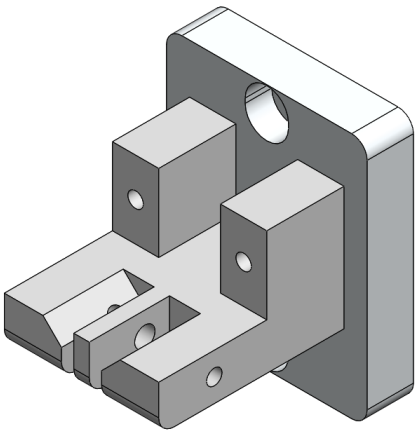


Figure 3.3: 3D CAD model of the designed assembly.

3.2 Software Implemented

An intelligent system requires a robust architecture. This architecture is built upon ROS, which is described in the topics that follow, in conjunction with the calibration

and visualization tools.

Robot Operating System (ROS)[1] is a framework developed by the Open Robotics foundation that allows an easier way to create software for robots. At its core, ROS is a collection of tools, libraries and conventions aiming to simplify the creation of complex and robust robot behaviors across a wide variety of robotic platforms. The hardware abstraction and the inter-communication between processes (called nodes) allow a truly collaborative development since a package designed for one project can easily be implemented in another one, even without the need of knowing exactly what happens inside the node. C++, Python and JavaScript are some of the supported languages. Nodes written in different coding languages can easily communicate between them in the ROS ecosystem.

ROS Industrial [2] is a software platform that builds upon ROS. This extension brings with it libraries, tools and drivers to communicate with industrial hardware.

MoveIt [3] has its foundation on previously existing robot movement frameworks, that were designed focused on the motion planning, trajectory generation and environment analysis for the PR2 arms. MoveIt accomplishes the separation of core algorithm capabilities from the middleware. This separation makes easier to reuse code by different manipulators. Figure 3.4 represents the MoveIt interaction between nodes/actions and in which topics they rely on.

On top of all this, Flexible Collision Library implementation allows for collision checked motion plans [Chitta *et al.* 2012].

[1] `https://www.ros.org/`
[2] https://rosindustrial.org/
[3] `https://moveit.ros.org/`

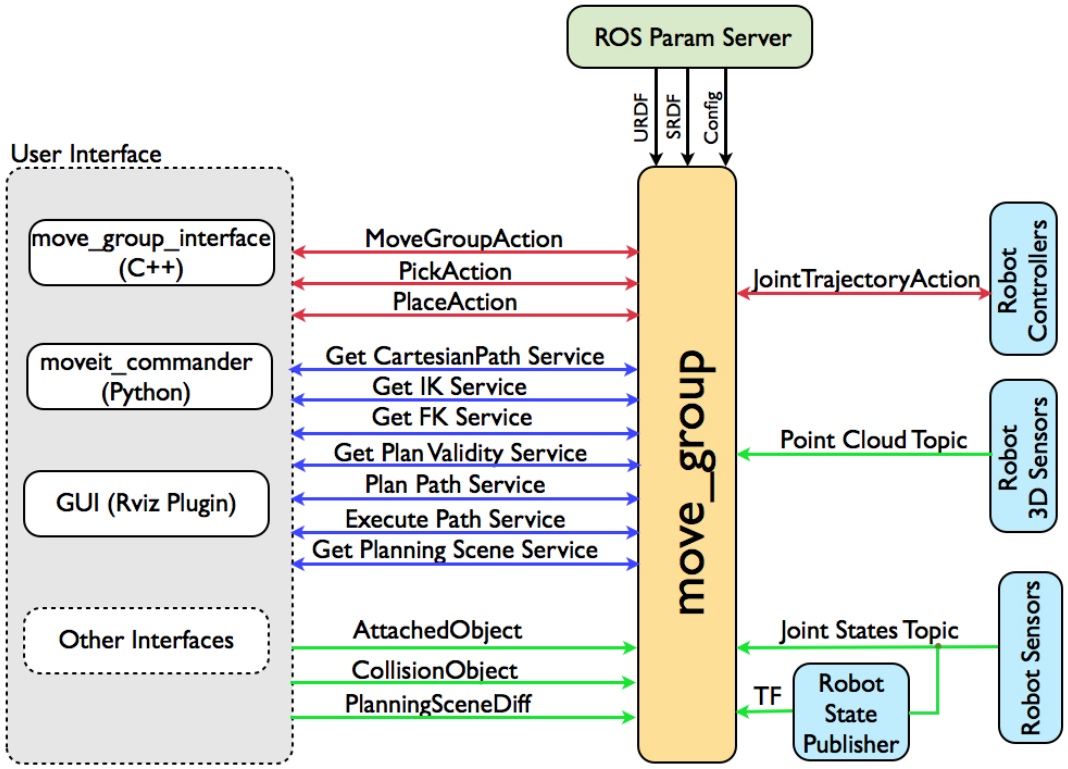


Figure 3.4: MoveIt system architecture [3].

OpenNI 2 (Open Natural Interaction 2)[4] offers a simple way to communicate with various camera sensors, specially RGB-D ones.

OctoMap Server [5] is a ROS node which implements OctoMap in such way that loads and distributes the built map to any other nodes that request it.

Vision ViSP (Visual Servoing Platform)[6] are a set of packages providing ViSP integration in ROS. ViSP is "a modular cross platform library that allows prototyping and developing applications using visual tracking and visual servoing technics".

Aruco ROS [7] is a ROS package which aims to the detection, recognition and pose estimation on Aruco markers. Aruco ROS also implements a new message type in ROS simplifying the transmission of data between the various nodes.

ARUCO / ViSP Hand-Eye Calibrator [8] package provides a simple integration between Aruco ROS and Vision ViSP that allows a simple camera pose estimation and calibration process. It can do two types of calibrations: eye-in-hand or eye-on-base, as discussed on 2.4.1.

[4] `http://wiki.ros.org/openni2_launch`
[5] `http://wiki.ros.org/octomap_server`
[6] `http://wiki.ros.org/vision_visp`
[7] `http://wiki.ros.org/aruco`
[8] `https://github.com/jhu-lcsr/aruco_hand_eye`

Hector Models [9] is package that provides multiple urdf models of sensors and components in the form of xacro files and meshes. This is useful mostly for visualization purposes but also for calibrating these sensors with any other exterior frame. In Fig. 3.5 is demonstrated the model of the camera used. Although the outer shape is not exactly as Fig. 3.2 suggested, the important part is that it has the exact measures between frames. Also, a simpler shape is faster for the collision check algorithm to process.

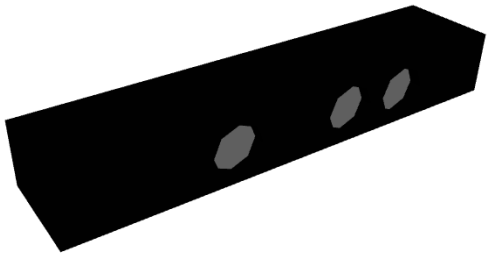


Figure 3.5: Asus Xtion model from Hector Models' package.

Camera Calibration [10] its a bridge between ROS image acquisition enabled devices and OpenCV camera calibration algorithms. This user intuitive software (represented in Fig. 3.6) allows for a easy gathering of frames containing a checkerboard, the computation of the best fitting calibration values and their export to the default directory for monocular or stereo camera setups.

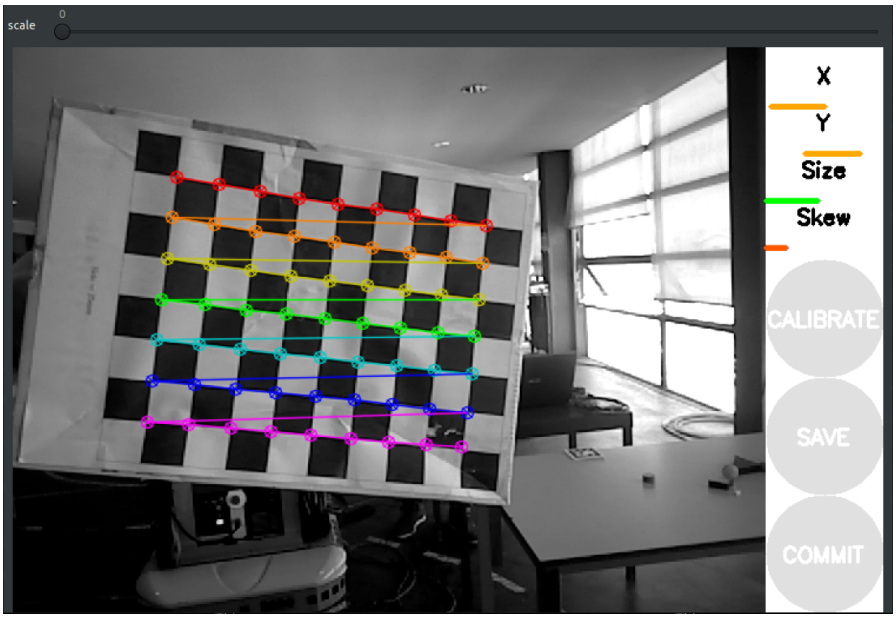


Figure 3.6: UI of camera_calibration for intrinsic calibration using a 9x7 checkerboard with size equal to 25 mm.

[9] https://github.com/tu-darmstadt-ros-pkg/hector_models
[10] http://wiki.ros.org/camera_calibration

Point Cloud Library (PCL)[11] is an open source library that mainly aims to point cloud processing. "From an algorithmic perspective, Point Cloud Library (PCL) is meant to incorporate a multitude of 3D processing algorithms that operate on point cloud data, including: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration, etc." [Rusu and Cousins 2011]. As an example, [Rusu and Cousins 2011] also present the flow chart of Fig. 3.7 that explains how object clusters are found on top a table, using PCL libraries.

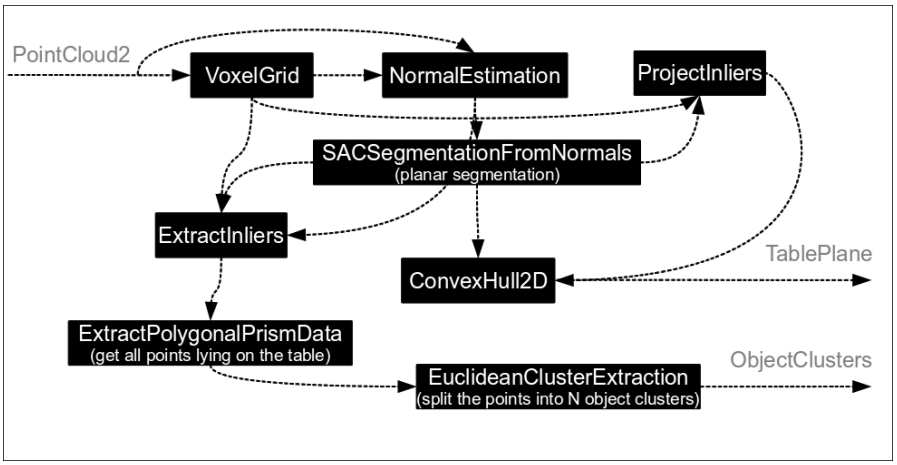


Figure 3.7: Example of an application of PCL structure to detect object clusters on a planar surface [Rusu and Cousins 2011].

Octomap Tools [12] are a collection of packages, originally developed by Professor Miguel Oliveira and Eng. Rafael Arrais [Arrais *et al.* 2017], that were further developed in this work. The main goal is to expand the already broad librarys of OctoMap and PCL with new tools, not only for developing but also debugging.

Now that it is clear the tools used in achieving the autonomous system we can proceed, in the next section, detailing the installation of ROS Industrial in the controller.

3.3 ROS Industrial

As stated on section 3.2, ROS Industrial, allied with the FANUC driver package, contains the drivers and messages requested to send commands to the manipulator. Yet some packages did not exist for the concrete one used and so adaptations were made.

3.3.1 Installation on the Controller

The robotic manipulator present at Laboratório de Automação e Robótica (LAR) did not have ROS Industrial installed. Without it, any ROS based communications were impossible, making it a crucial first step.

[11] `http://www.pointclouds.org/`
[12] `https://github.com/miguelriemoliveira/octomap_tools`

The source files are distributed by the official ROS Industrial repository[13]. After compiling them in RoboGuide (the official robot simulator software of FANUC) to the right controller version, the files where loaded using a USB drive. This source files are written in KAREL and are specific for the FANUC controllers.

Some configurations where also needed. Since the *ros_ state* and *ros_ relay* programs make use of User Socket Messaging, they expect two server tags to be available, and so, they need to be configured. A couple of flags, integer and position registers are also needed in order to *ros_ relay* and *ros_ movesm* work properly and must be set.

Last steps were to implement the default configurations for both *ros_ relay* and *ros_ state* as described in tables 3.1 and 3.2. Lastly, the TPE program *ros_ movesm* needed to be updated to use the newly configured flags.

Table 3.1: Default configuration of *ros_ relay*[14].

Name	Type	Default	Unit	Description
checked	BOOLEAN	False	-	Configuration has been completed by user
f_msm_rdy	INTEGER	1	-	movesm i'face: 'ready/ack' signal flag
f_msm_drdy	INTEGER	2	-	movesm i'face: 'data ready' signal flag
loop_hz	INTEGER	42	Hz	Main loop update rate
move_cnt	INTEGER	50	%	CNT to set with each joint motion instruction
move_speed	INTEGER	20	%	Joint speed to set for all trajectory points
pr_move	INTEGER	1	-	movesm i'face: position register for next trajectory point
r_move_spd	INTEGER	1	-	movesm i'face: integer register for motion speed
r_move_cnt	INTEGER	2	-	movesm i'face: integer register for CNT value
s_tcp_nr	INTEGER	11000	-	TCP port to listen on
s_tag_nr	INTEGER	4	-	Index of the Server Tag to use
um_clear	BOOLEAN	True	-	Clear user menu on start

[13] `http://wiki.ros.org/fanuc` accessed on March 12, 2019.

Table 3.2: Default configuration of *ros_ state*[14].

Name	Type	Default	Unit	Description
checked	BOOLEAN	False	-	Configuration has been completed by user
loop_hz	INTEGER	42	Hz	Main loop update rate
sloop_div	INTEGER	10	-	Divider for robot_status reporter loop
s_tcp_nr	INTEGER	11002	-	TCP port to listen on
s_tag_nr	INTEGER	3	-	Index of the Server Tag to use
um_clear	BOOLEAN	True	-	Clear user menu on start

In the end of all configurations, the "checked" setting of both tables 3.1 and 3.2 must be set to true in the corresponding programs.

3.3.2 Creation of the FANUC M-6iB/6S Package

To receive the joints state and, even, visualize the robot in Rviz, some packages were needed that were not available by the official providers for the model used in this dissertation. Yet, the variant with a slightly larger arm (M-6iB) was available. This was the base for the adaptation done.

The first and most important step was to update the xacro file. This file has all the mechanical details (link lengths, joint limits, joint velocities, etc.) of the given robot (see table 3.3), and will be used in the generation of the Unified Robot Description Format (URDF). Based on the M-6iB Series manual [FANUC 2007] the link measures were adapted. Only links two and four required dimension changes, as demonstrated by Fig. 3.8.

[14] `http://wiki.ros.org/fanuc/Tutorials/hydro/Configuration` accessed on March 16, 2019.

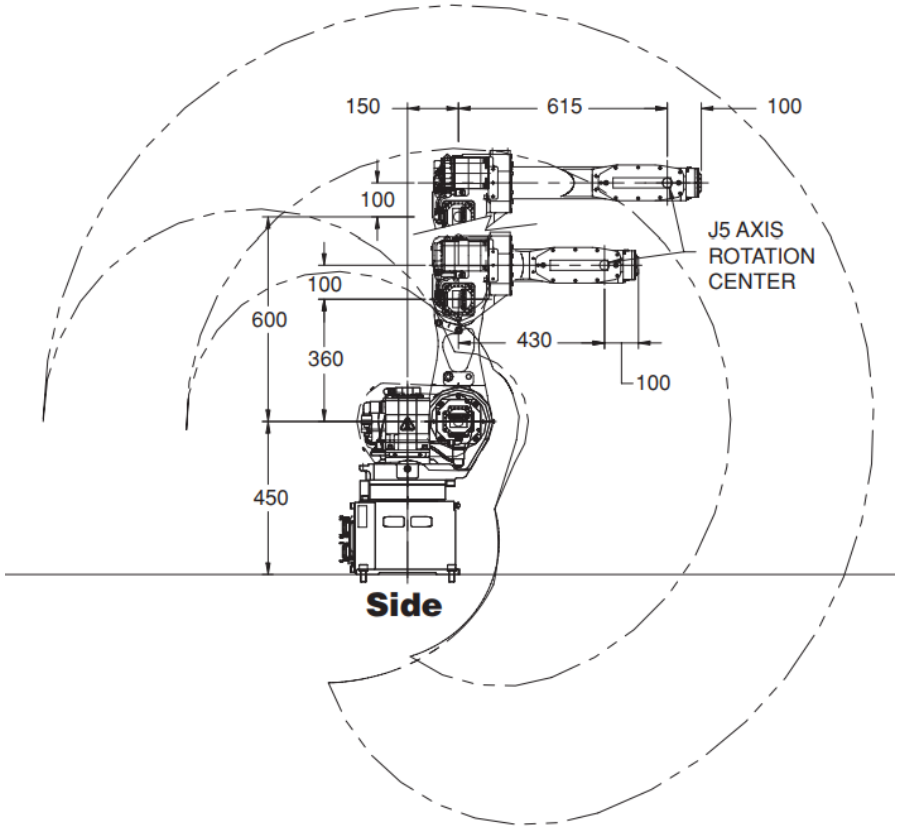


Figure 3.8: Comparison of the dimensions between the M6iB and M6iB/6S. Adapted from [FANUC 2007].

Motion speeds and ranges had to be adapted has well. They are presented on table 3.3 and were also taken from [FANUC 2007].

Table 3.3: Mechanical properties of the M-6iB Series [FANUC 2007].

M-6*i*B Series Specifications

Items		M-6*i*B	M-6*i*B/6C Cleanroom	M-6*i*B/6S 6 kg Payload[7]	M-6*i*B/6S 10 kg Payload[7]	M-6*i*B/2HS	M-6*i*B/2HS Solution Arm
Axes		6	6	6	6	6	6
Payload at Wrist (kg)		6	6	6	10	2	2
Supplementary Payload on J3 Casting[1] (kg)		12	12	12	6	12	12
Reach (mm)		1,373	1,373	951	951	951	951
Repeatability[2] (mm)		± 0.08	± 0.08	± 0.08	± 0.08	± 0.08	± 0.08
Interference Radius (mm)		273	350	273	273	273	273
Motion Range[3] (degrees)	J1	340	340	340	340	340	340
	J2	250	250	250	250	250	250
	J3	315	315	310	310	310	310
	J4	380	380	380	380	380	240
	J5	280	280	280	280	280	110
	J6	720	720	720	720	720	720
Motion Speed[4] (degrees/s)	J1	150	150	200	200	200	200
	J2	160	160	200	200	200	200
	J3	170	170	260	200	260	260
	J4	400	400	400	400	400	400
	J5	400	400	400	400	400	400
	J6	520	520	720	520	1,200	2,000
Wrist Moment (kgf-cm)	J4	160	160	160	160	160	140
	J5	100	100	100	160	100	80
	J6	60	60	60	70	20	12
Wrist Inertia (kgf-cm-s^2)	J4	6.4	6.4	6.4	6.4	6.4	6.2
	J5	2.2	2.2	2.2	3.7	2.2	2.0
	J6	0.62	0.62	0.62	0.65	0.36	0.20
Mass (kg)		138	145	135	135	135	140
Mounting Positions		Floor, angle[5], wall[5], inverted	Floor, inverted	Floor, angle, wall, inverted	Floor, angle, wall, inverted	Floor, angle, wall, inverted	Floor
Vibration (m/s^2)		≤4.9 (0.5G)	≤4.9 (0.5G)	≤4.9 (0.5G)	≤4.9 (0.5G)	≤4.9 (0.5G)	≤4.9 (0.5G)
Special Application Environments[6]		IP67 forearm and wrist / IP54 lower body	ISO Class 5 (Class 100 per U.S. Federal Standard 209E)	IP67 forearm and wrist / IP54 lower body	IP67 forearm and wrist / IP54 lower body	IP67 forearm and wrist / IP54 lower body	IP67 forearm and wrist / IP54 lower body (No IP rating for solenoid box)

In addition to the xacro file, ROS requires the meshes of the links for collision checking. Yet, as discussed earlier, this exact model did not had its implementation, but the slightly longer version had. To have a CAD of the M6iB/6S manipulator, with the right geometry and dimensions, the CAD from the longer M6iB had its links two and four shortened to accommodate the actual dimensions. As result, the CAD is now compliant with the actual dimensions of M6iB/6S. The difference between the original and adapted meshes can be seen on Fig. 3.9.

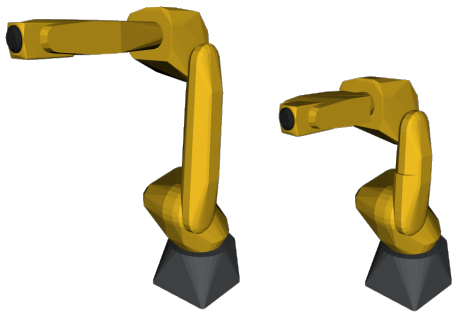


Figure 3.9: Result achieved with the described implementation, on the left the original M-6iB and on the right the modified version, the M-6iB/6S.

Last step was to create the *moveit_ config* package using the setup assistant (see Fig. 3.10).

All this changes resulted in a creation on two packages that were submitted to the official repository.

3.4 The FANUC Xtion Package

As introduced in 3.3.2, the integration with MoveIt requires an extra package that is created with *MoveIt Setup Assistant* (as seen in Fig. 3.10).

MoveIt Setup Assistant is a tool for configuring any robot for use with MoveIt, that generates a Semantic Robot Description Format (SRDF) file for the desired robot, alongside others necessary, used on the pipeline. This other files consist in various launch and configuration files with information, for example, about which pairs of links collision checking is not needed and the definition of the kinematics chain.

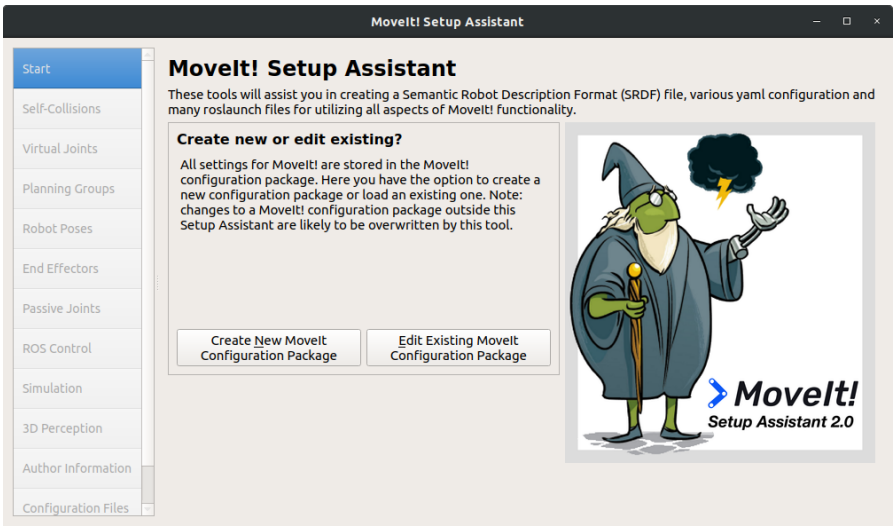


Figure 3.10: MoveIt Setup Assistant UI.

Before testing in the real manipulator, a FANUC specific simulation software – *RoboGuide* –, was used to verify all these adaptions.

3.5 RoboGuide Testing

To test all the packages created and adapted until this point, a simulation was carried out using RoboGuide (shown in Fig. 3.11). The connection is pretty much straight forward, the ethernet cable is connected in both machines, their corresponding IP's are configured and when launching the ROS nodes, the IP of the client is passed as an argument.

In RoboGuide the manipulator used was exactly the same but the controller version was v7.70 that was more recent than the v7.20 present on the real controller. This procedure proved that the packages were correctly built and that the real controller needed a software update in order to be possible the ROS Industrial implementation as described in section 3.3.1.

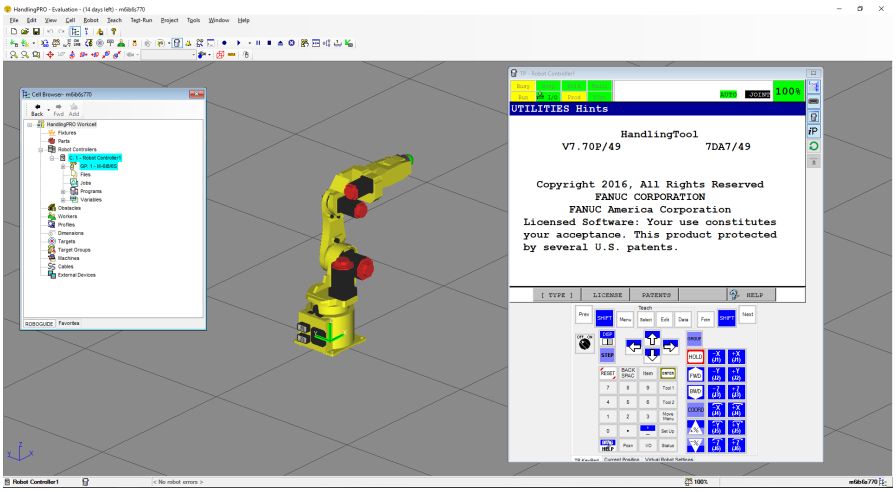


Figure 3.11: Roboguide UI with the FANUC M6iB/6S.

3.5.1 Updating the Software on the Controller

The FANUC driver package was developed to work with robot's controller version v7.70 and above, and so a software update was needed. Actually, the word update is a bit of a stretch since the actual procedure consists in copying a system image from version v7.70 and uploading it to the controller. This update was executed by Motofil Robotics, S.A., who brought an already created image file for version v7.70, with all the programs and configurations coming from the LAR's v7.20 backup.

After checking that everything did update correctly, it was missing the mastering of the manipulator. The mastering of the robot is like its calibration. All joints are manually rotated to their zero configuration (with the help of markers, seen in Fig. 3.12, which allow the correct joint alignment) and, in the end, the controller knows that, from that point on, every joint displacement value is taken in reference to that configuration.

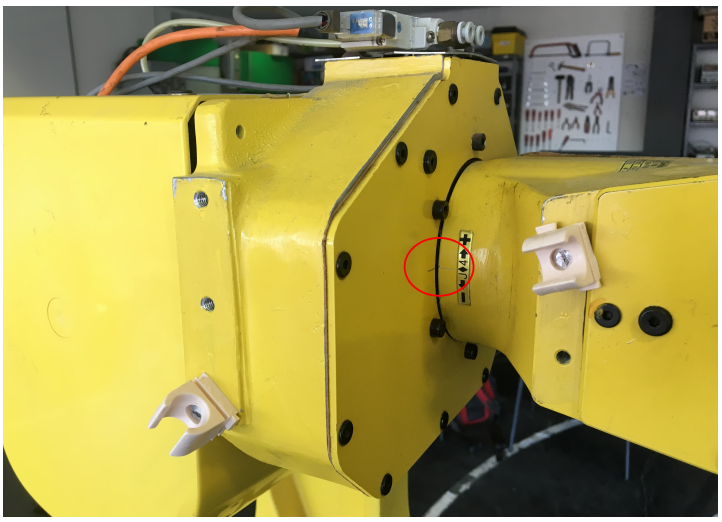


Figure 3.12: Mastering mark in joint 4 of the FANUC M6iB/6S.

After the whole process, the manipulator was finally able to execute motion requests coming from MoveIt planner, exactly like happened within RoboGuide simulation.

3.6 Calibration

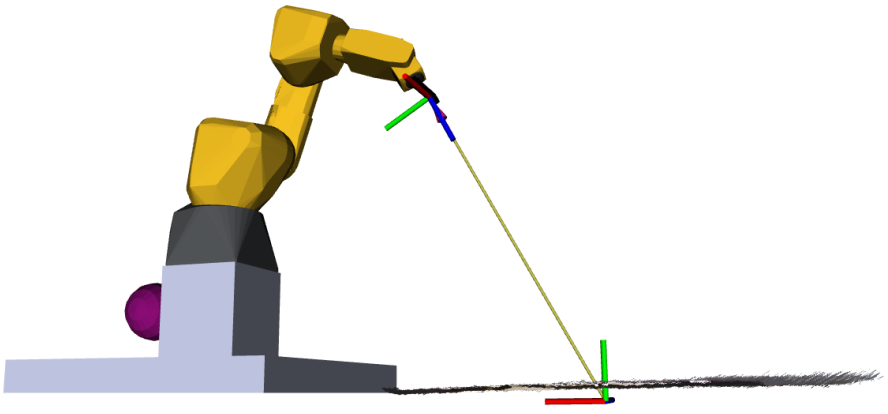
The calibration procedure went out in two parts: i) the RGB camera intrinsic parameters calibration and ii) the extrinsic calibration as well. Both were important to be obtained, given that what the RGB camera perceives will influence the outcome of the extrinsic calibration. This connection between both types of calibration is established in the next section.

3.6.1 Intrinsic Calibration

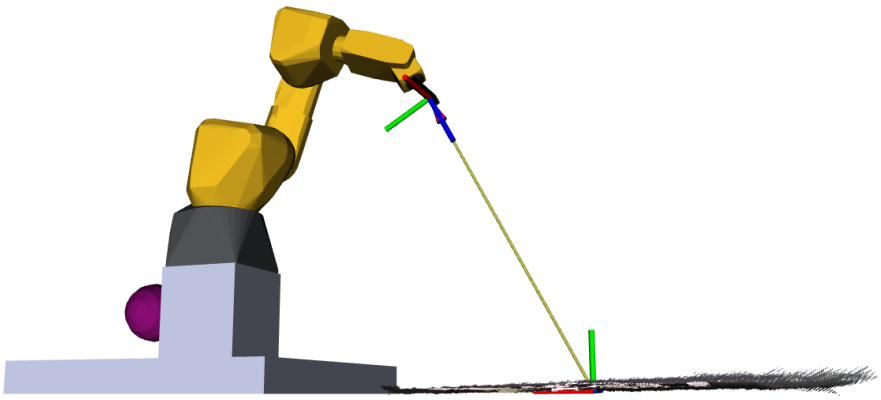
There are two ways to measure distance with a RGB-D sensor: using the RGB image and a pattern with known dimensions to estimate it or using the depth sensor that, for each ray, does triangulation to take the measure.

Obtaining the pose of the camera relative to a frame using the point cloud data is not trivial given that a point (or a set of points) do not contain information about orientations. The most common way is to detect a known marker, as already introduced. This detection returns the pose of the camera relative to the marker reference frame. In the case where the RGB sensor wrongly estimates its pose, it induces errors in the computation of the geometric transformation between the end effector and itself (extrinsic calibration). Hence, the need for intrinsic calibration.

Figure 3.13 shows the improvements made after the intrinsic calibration process. Before it (Fig. 3.13a) the height estimates obtained by the RGB and depth sensors were inconsistent. Following the obtainment of these parameters (Fig. 3.13b) these estimates are more precise.



(a)



(b)

Figure 3.13: Difference between an uncalibrated system (a) and a calibrated one (b). Distance between the marker pose and the point cloud reduces significantly in the calibrated system.

This calibration is done showing multiple and representative (this is, they are not redundant) shots of a checkerboard pattern, with known dimensions, to the camera (an example is demonstrated on Fig. 3.14). When enough samples are gathered, the algorithm calculates the best fitting values (see listing 3.1) for the focal length, image sensor format and principal point, as well as distortion, rectification and projection coefficients.

Figure 3.14: Example of the calibration procedure. The checkerboard has 8 x 6 interior corner edges with each square side measuring 0.105 m.

```
image_width: 640
image_height: 480
camera_name: rgb_PS1080_PrimeSense
camera_matrix:
    rows: 3
    cols: 3
    data: [535.984359806132, 0, 317.7452534492942, 0, 536.4898690073841,
    ↪  241.8439551122369, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
    rows: 1
    cols: 5
    data: [0.0806854339221736, -0.07987970254488948,
    ↪  -0.002330483620940096, -0.006520012903618213, 0]
rectification_matrix:
    rows: 3
    cols: 3
    data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
    rows: 3
    cols: 4
    data: [547.8181762695312, 0, 313.5475346711901, 0, 0,
    ↪  551.4313354492188, 240.7256027340391, 0, 0, 0, 1, 0]
```

Listing 3.1: Example of an intrinsic calibration yaml file.

After this process is over, it is now possible to extrinsically calibrate de robot with the confidence on the frames coming from the camera. This next step is described in section 3.6.2.

3.6.2 Hand-in-Eye Calibration

Extrinsic calibration (from now on called hand-in-eye calibration, for being the method used) is the operation which returns the geometric transformation between two, rigidly attached, frames that previously was not known. This is crucial to have in order to correctly record the information, coming from the camera in any arbitrary pose, in the world's reference frame.

The ARUCO / VISP Hand-Eye Calibration[15] provides an easy way to implement Aruco based extrinsic calibration to the system. By identifying the Aruco position relative to the camera's RGB sensor, and knowing the transformation between the base link and the end-effector, it can infer the static transformation between this last link and the camera itself, based on what described on section 2.4.1.

[15] `https://github.com/jhu-lcsr/aruco_hand_eye` accessed on march 16, 2019.

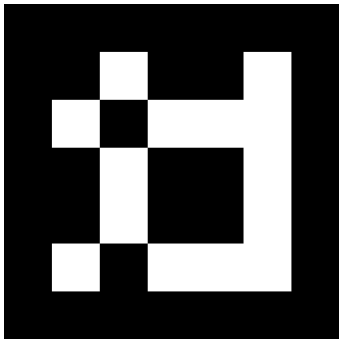
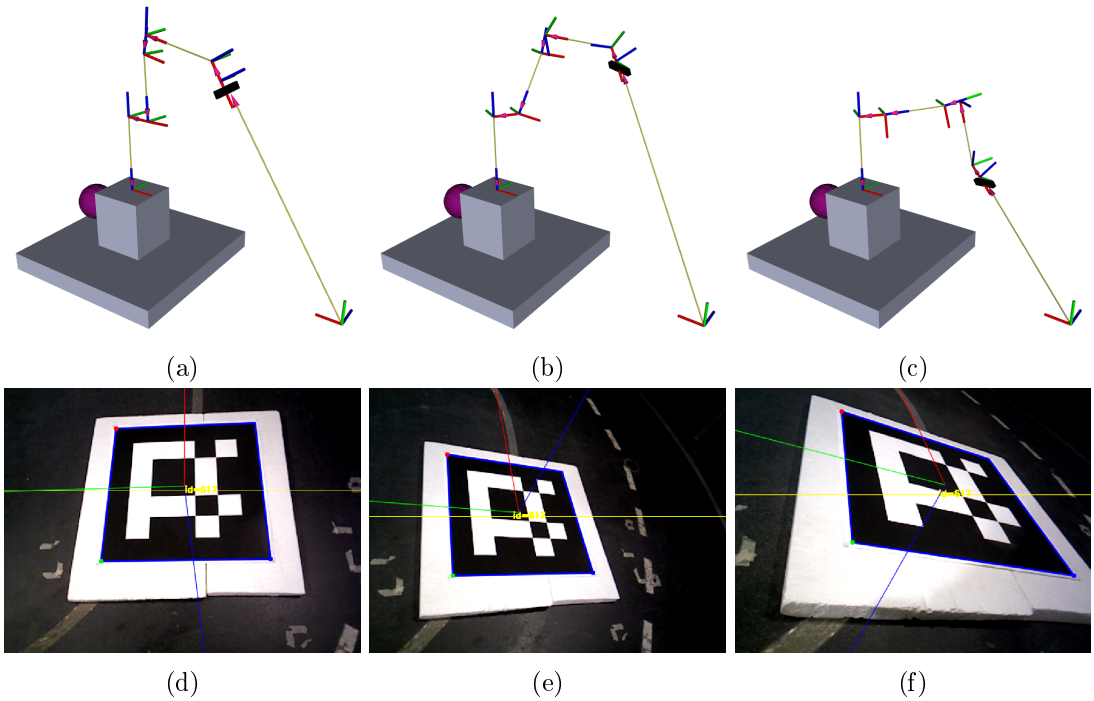


Figure 3.15: Aruco marker (original dictionary, ID 617) used in hand-in-eye calibration.

In the implementation proposed, the calibration mode assumes the existence of two apparatus on the environment, the manipulator and the camera, since they are, in fact, to different peaces of hardware that, at this point, have no information of their positioning relative to each other. Each one has its own parent frame: the manipulator's is assumed to be the world reference frame. In the other hand, the camera's reference frame will be rigidly attached to the end effector. The goal here is to move the manipulator in ways which the camera can detect the Aruco (observing it from different poses, see Fig. 3.16), so it progressively tends towards a geometric transformation that fits the best with the real world assembly. When the calibration reached a point where it is considered good, those parameters are stored.

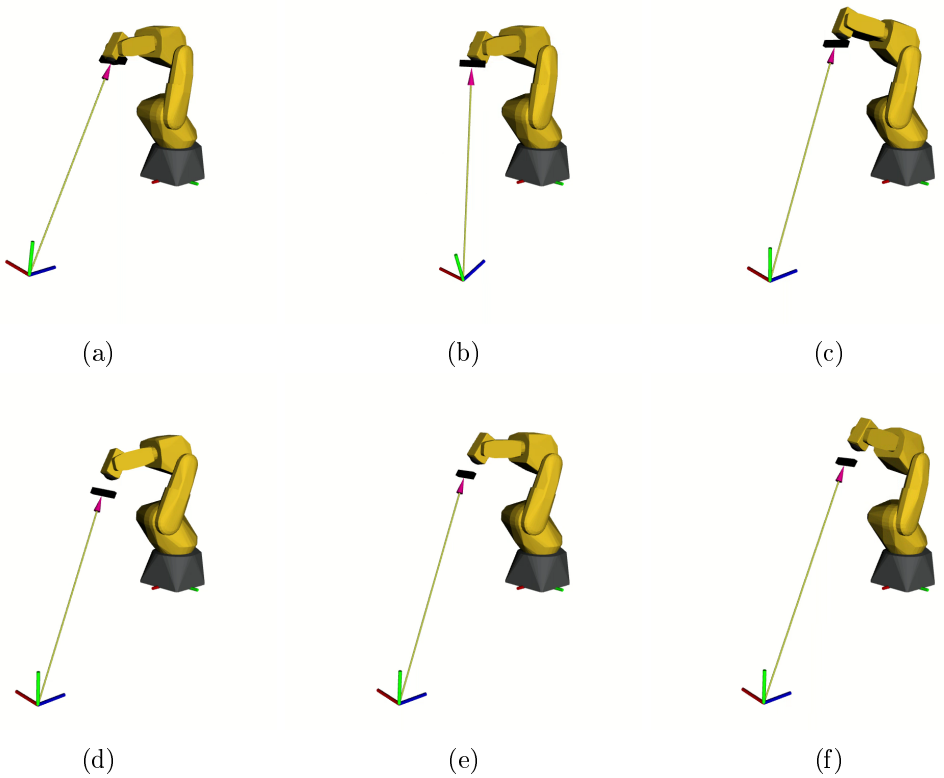


(a) (b) (c)

(d) (e) (f)

Figure 3.16: Aruco detection for three different camera poses. (a), (b) and (c) show the frames corresponding to each joint, the camera and where it measures to be the marker. (d), (e) and (f) are the respective images that originate those detections.

Figure 3.17 details, in a temporal line, the evolution of the process. The first time it detects the marker (3.17a) the camera is, expectably, misplaced to the point that, the first few collected samples, even change where the system detects the marker, relative to the manipulator (Fig. 3.17b). Successively giving the algorithm more samples (by changing both position and rotation) improves the result, moving the camera model towards the real position, ending up like in Fig. 3.17f.



(a) (b) (c)

(d) (e) (f)

Figure 3.17: Improvements on the extrinsic calibration of the camera relative to the end effector. Temporal line is from (a) to (f). The frame with an arrow to the camera is where the Aruco marker is being detected.

With the hand-in-eye calibration completed the obtained parameters are saved in a specially created xacro file, exactly like in listing 3.2, as well as the transformations tree on appendix B. This makes it possible to integrate, in one single robot, both the camera and the manipulator, by creating its description, as discussed further ahead.

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
<!--This xacro contains six xacro propperties representing a
↪  transformation obtained through calibration-->
<!--Calibration generated at 14:11:10, 22/05/2019 -->
<!--This file was written automatically using the "rosrun
↪  smobex_calibration store_calibration" command. -->
<xacro:property name="roll" value="0.0705631628597"/>
<xacro:property name="pitch" value="0.0800765806867"/>
<xacro:property name="yaw" value="0.00297065394398"/>
<xacro:property name="x" value="0.108546244873"/>
<xacro:property name="y" value="0.00786073211074"/>
<xacro:property name="z" value="0.00438827570731"/>
</robot>
```

Listing 3.2: Example of a calibration xacro file.

3.6.3 Robot Description

Contrary to the calibration procedure, the operation mode knows the existence of only one robot, that is the coupling between manipulator and camera. This coupling is possible based on the previously saved calibration.

The description file for this robot imports both already existent FANUC M6iB/6S and Xtion camera description files and creates a joint connecting both. This joint has the same exact parameters of the calibration by also importing the file generated in that step.

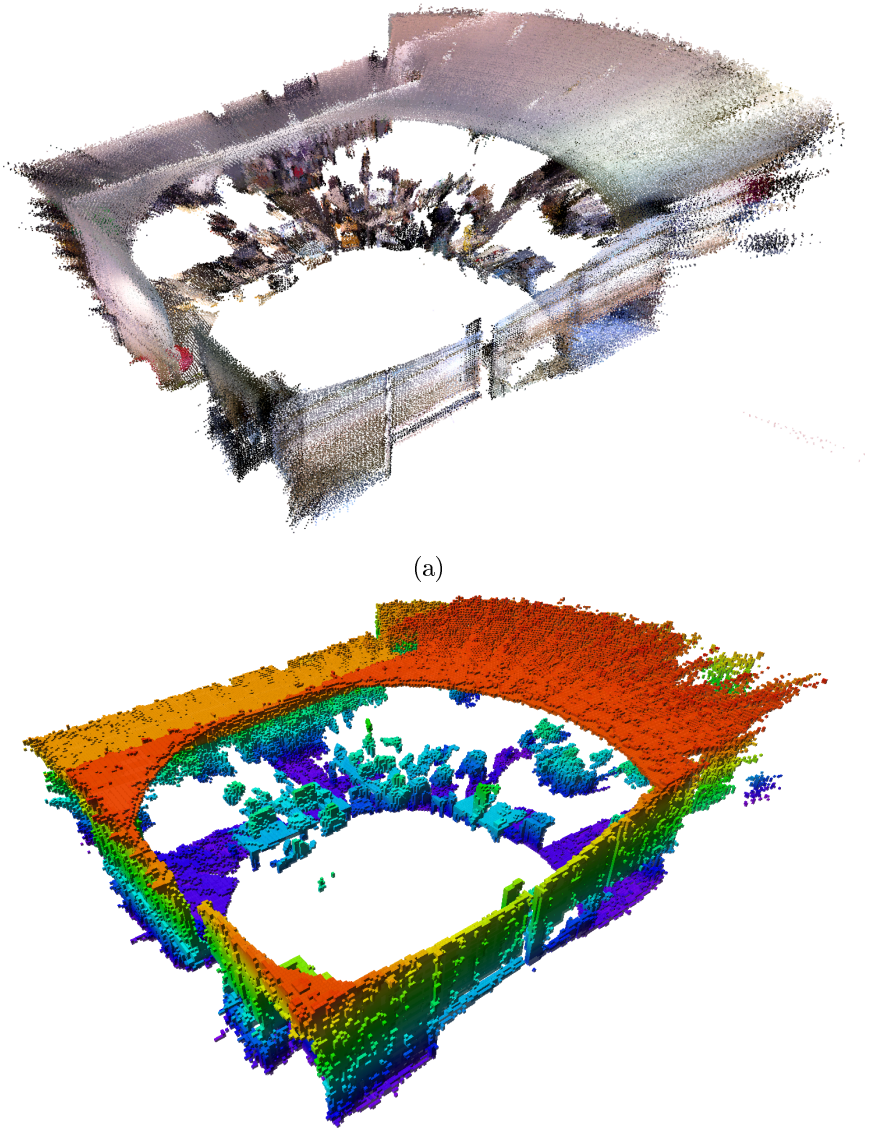
Figure 3.18: The fully calibrated robot visualized in Rviz.

If the hand-in-eye calibration resulted in an accurate model, this robot can now provide correct transformations between any links that compose it, becoming the base for all the work described from this point on. To infer, in an qualitative manner, the quality of the calibration, an accumulation of various point clouds is performed.

3.7 Accumulation of Point Clouds

Sometimes is useful to have a tool that records and stores an accumulation of point clouds and the corresponding OcTree representation. Calibration checking is one of that times.

Figure 3.19 has both previously mentioned types os representation of LAR which, for reference, has 17.4 m × 9.3 m × 2.7 m (length × width × height). Those representations where used to validate the performed calibration.



(a)

(b)

Figure 3.19: Reconstruction of LAR using a point cloud (a) and the corresponding OcTree map (b).

With this data the user can, qualitatively, check if the calibration procedures resulted in acceptable values or if something went wrong and exists the need to redo the calibrations.

Now that there is the certainty of a reliable robot model, the missing piece for reaching the end goal – a robotic manipulator with autonomous exploration capabilities – is the intelligence behind the decision making.

3.8 Defining the Exploration Volume

The objective of the exploration may not be to explore the entire room but, most probably, just a small portion of it, as defined in the example of Fig. 3.20. By defining that volume, the algorithm ignores everything outside it and only whats inside matters. To generate de Octree map the points outside the bounding box are filtered and not passed to OctoMap, so its just like if they were not sensed.

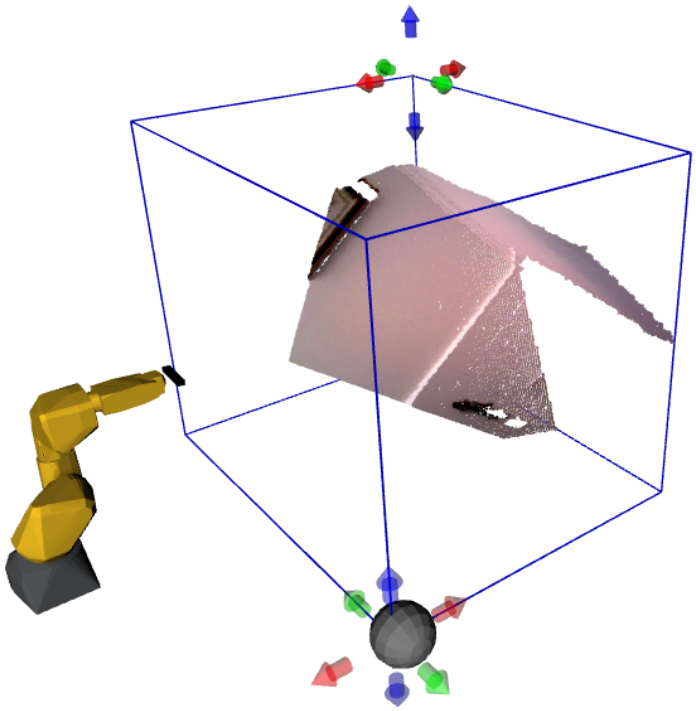


Figure 3.20: Exploration volume definition in Rviz.

Effectively there are two box regions delimiting the environment (see Fig. 3.21d): i) Point Cloud Filter Volume, and ii) Exploration Volume. Exploration Volume is defined by the user and it is the volume that will be explored. Point Cloud Filter Volume is a user-defined scale of the first one in which the point cloud will be recorded (see Fig. 3.21). This ensures two things: i) the OctoMap does not becomes huge and detailed in places where it does not need to be and ii) that there is not an exaggerated amount of information filtered out.

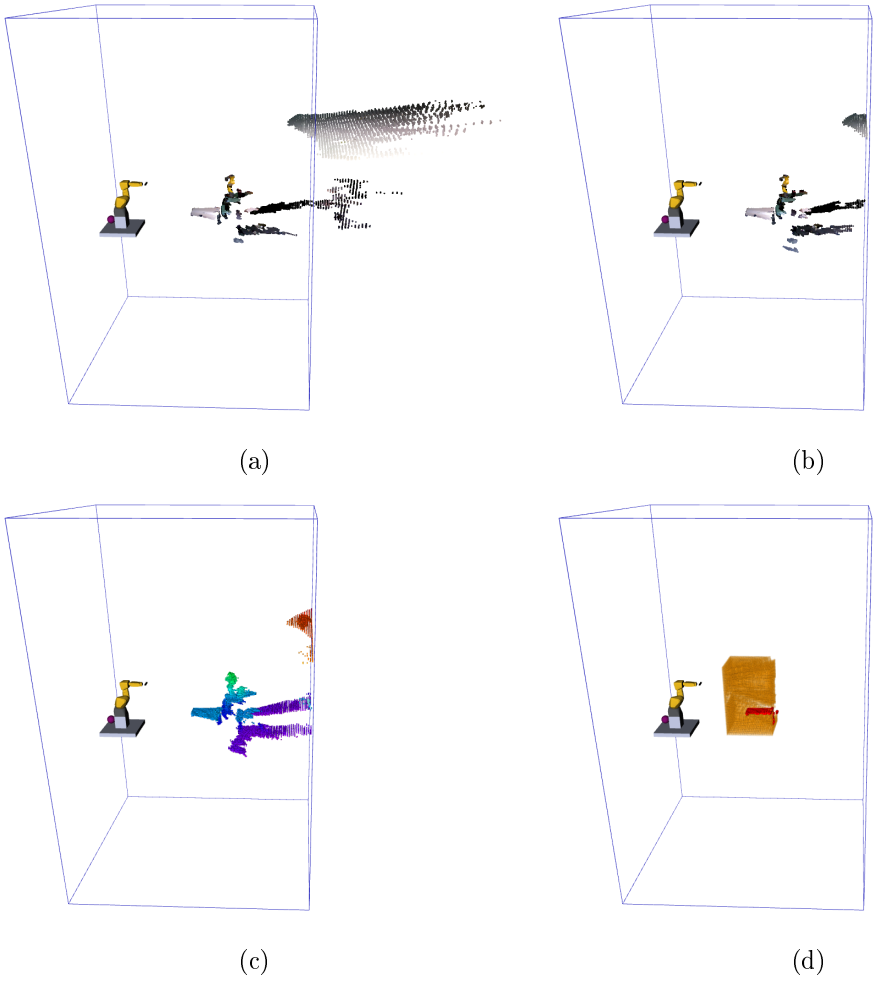
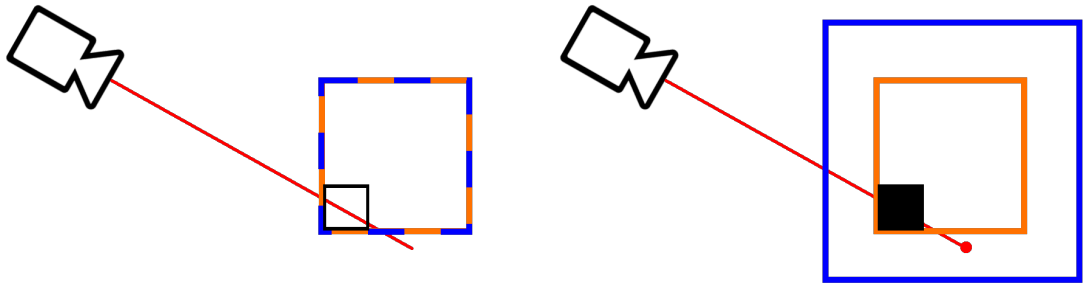


Figure 3.21: Definition of the point cloud filter volume (each side is five times bigger than those of the exploration volume). a) Full point cloud coming from the camera. b) Filtered point cloud from (a). c) Generated OctoMap based on the filtered point cloud (free space hidden for visualization purposes). d) Exploration volume (Orange represents unknown space, red occupied space. Free space hidden for visualization purposes).

Taking a look at Fig. 3.22, the reason why we need the two bounding boxes is understandable. In this case, the orange square represents the exploration volume (in 2D for simplification) and the blue one is the point cloud filter volume: points outside it are not used in the OctoMap construction.

Because, in Fig. 3.22a, the search and filter lines are coincident, the beam bounces back outside the filter, disabling its registration, making it impossible to mark the black voxel as free. On the other hand, in Fig. 3.22b, the point is detected since its inside the point cloud filter, allowing the unveiling of the black voxel state.



(a) (b)

Figure 3.22: Visual example of the need to have two bounding boxes (here in 2D for simplification). In Fig. (a) they are coincident and since the ray is reflected outside the point cloud filter volume, it is impossible to know in which state the black voxel is. In (b) they are no longer coincident so the point is registered and the state of the voxel is now known.

A practical example of coincident point cloud filter and exploration volumes is shown in Fig. 3.23.

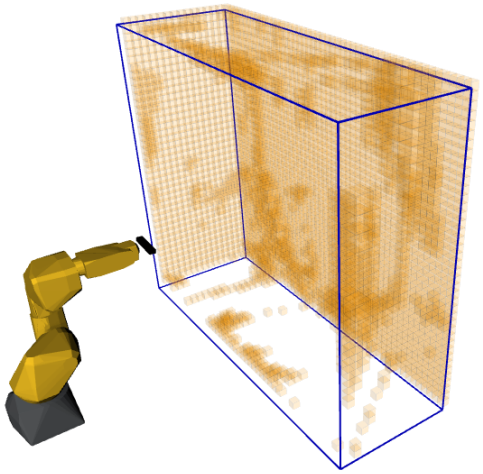


Figure 3.23: Example of space that cannot be defined (in orange) because the volumes are coincident.

A goal for the exploration, this is, the volume that is requested the robot to explore, has been established. Thus, the point cloud filter volume is defined as well, limiting the registered data points and, therefore, limiting the OctoMap build up. At a low resolution level, the voxels are all defined but just the ones that have been measured are explicitly stored.

3.9 Finding Unknown Space

Once that both exploration and point cloud filter volumes are defined, we need to assess the areas where there was not been gathered information yet. However, OcTrees, explicitly, does not store information about unknown voxels. This is a demand in this dissertation, considering that our proposed approach is established to give higher priority to poses which evaluate a greater amount of unknown voxels.

To extract this information, we iterate through all voxels of the OcTree and check their state. If is undefined then that voxel is still unknown. The center point of this unknown voxels are added to a newly constructed OcTree to take advantage of OctoMap's optimizations, decreasing the number of voxels that are published without losing information.

After optimizing the OcTree structure (which is, essentially, pruning it), its voxels are published for visualization (see Fig. 3.24). Also this newly created map is published as an OctoMap message where the voxels marked as free are equivalent to the ones that are unknown is the original map.

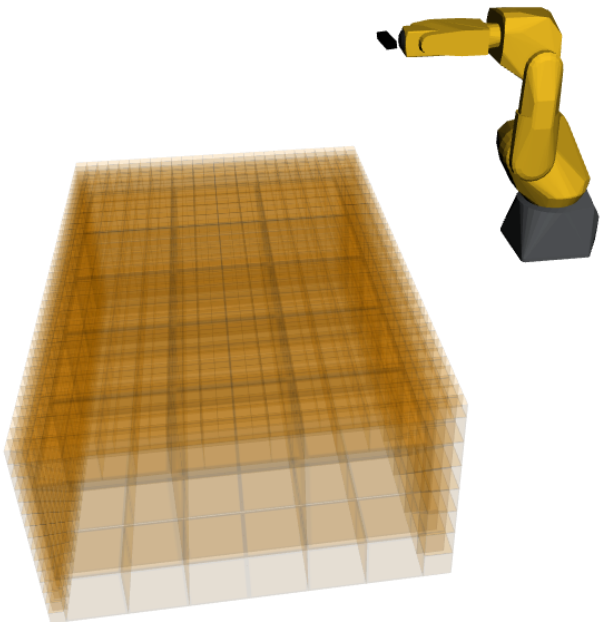


Figure 3.24: Visualization of the unknown space. Voxels representing unknown space are marked as translucent orange.

Considering that, after some evaluation iterations, the unknown space starts to be-

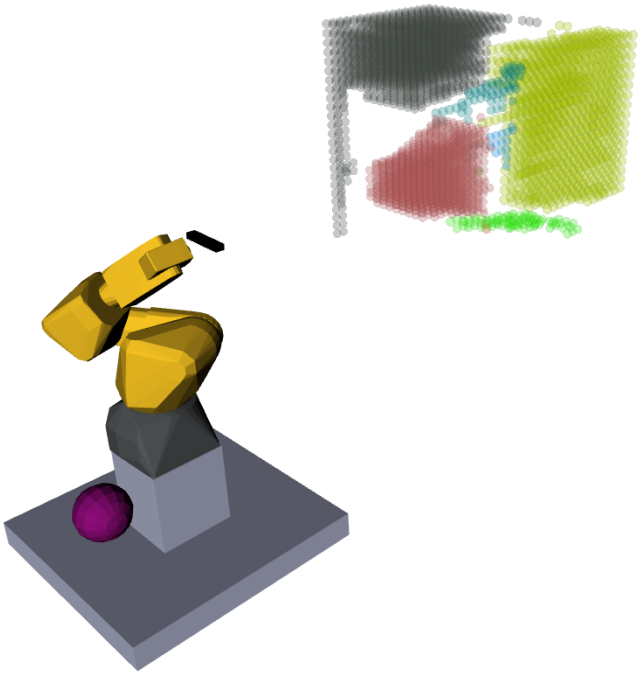
come broken (this is, clusters of unknown space start to form), we use this phenomenon in our advantage.

3.10 Finding Unknown Clusters

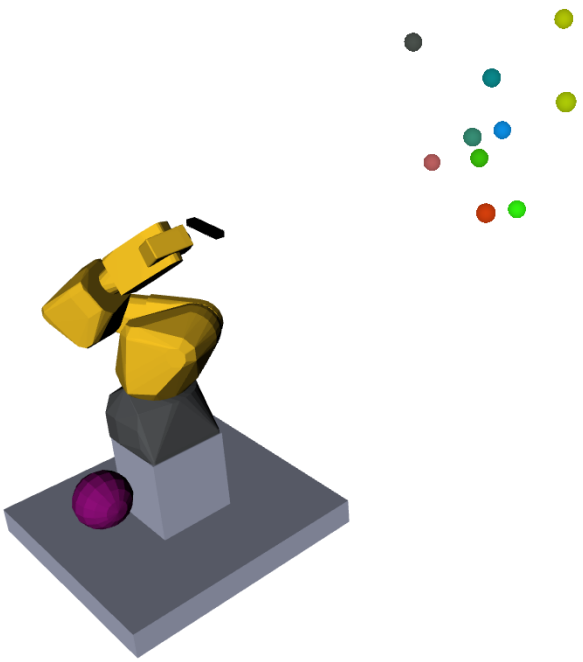
The approach taken subdivides the volume that still as no information about into clusters, so it can, further ahead, sample more plausible poses looking toward different volumes with the certainty that all of them are unknown.

Having the set of voxels that are unknown gives the possibility to compute every single center point if its constituent voxels, essentially giving a point cloud representative of the unknown space. Iterating through all these points, it can be tested if the distance to the neighbors is within a defined distance (in this case, that distance is just over the double of the OctoMap's resolution). If they are, in fact, close enough, they form a cluster and start growing until all its points neighbors also belong to the cluster or are too far to belong. At this point a new cluster formation starts and the process repeats itself until all the input points are tested. A result of this procedure can be observed in Fig. 3.25a. Notice that if a point is far enough of all others, it is also considered a cluster. The expected behavior is that the poses generated to observe it will have a low score and so a pose looking towards another cluster is chosen.

Knowing which points compose which clusters, the task of computing the centroids (the points that will define the generated poses' orientation) becomes trivial. Figure 3.25b shows the centroids corresponding to the clusters of Fig. 3.25a.



(a)



(b)

Figure 3.25: Example of ten clusters found after an exploration iteration. (a) Point clouds (points are the center points of the unknown voxels) that define each cluster. Different colors represent different clusters. (b) Respective clusters centroids.

The computed centroids are a requested information for the proposed pose sampling method.

3.11 Interactive Manipulator Movement

The ROS Industrial, MoveIt and the FANUC driver provide all the launch files and nodes that allow the movement of the manipulator end frame (camera_depth_optical_frame) with an interactive marker, and subsequently plan and actually move it. This is quite helpful for both manual and auto pose evaluation. When manually evaluating a desired pose the camera's depth optical frame is dragged to the desired position following the evaluation. This is shown in Fig. 3.26.

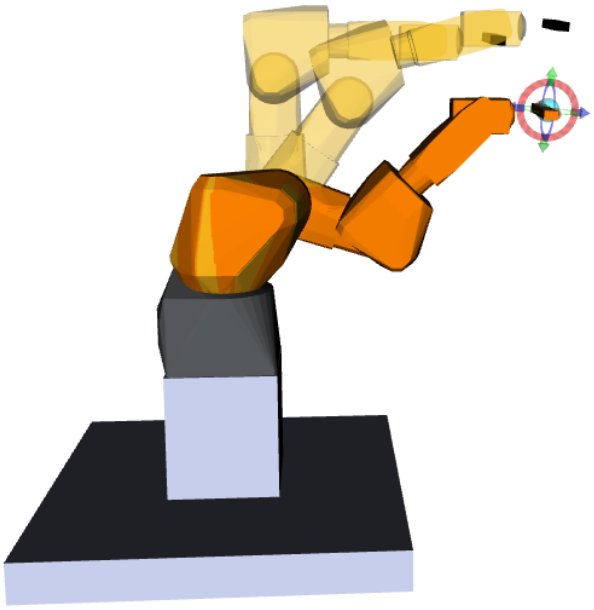


Figure 3.26: Rviz visualization of the robot planed path from one configuration to another. The end configuration is in darker, less transparent orange.

This feature, allied with the evaluation methods also used for the autonomous intelligence, provide an interactive way of manually test poses.

3.12 Autonomous Exploration Mode

Providing a way to explore a given volume autonomously is the end purpose of this work. In order to do so, its proposed the architecture of Fig. 3.27. This architecture allows for a fully autonomous exploration, that always guarantees the manipulator's movement to the pose (from the sampled set) predicted to give the most information gain, which it can plan for. The full node graph is available on appendix C.

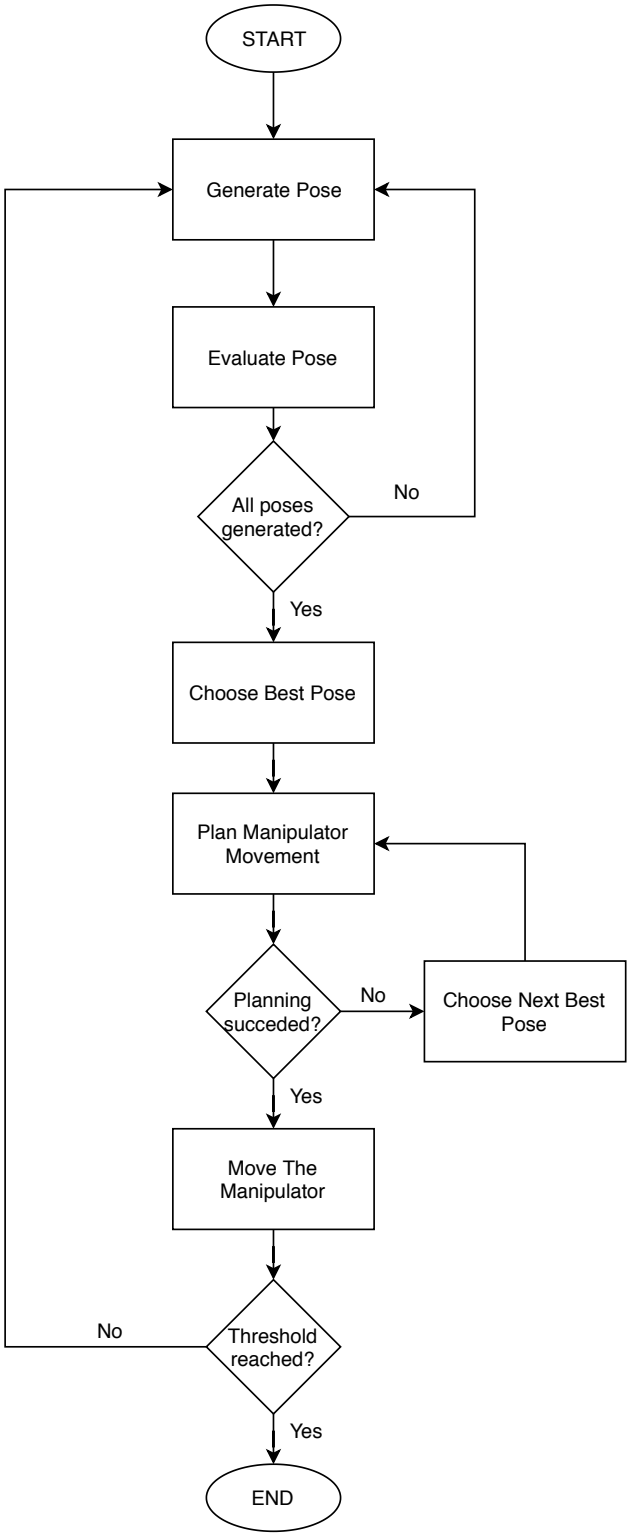


Figure 3.27: Flowchart of the full exploration algorithm.

3.12.1 Pose Sampling

Pose sampling requires information about the center of the clusters, so they acquire the best orientation. The total number of poses is a parameter and the algorithm evenly distributes that quantity by all the found clusters, as in eq. 3.1,

$$ppc = \text{round}\left(\frac{N_{poses}}{N_{clusters}}\right), \tag{3.1}$$

where ppc is the number of poses per cluster, N_{poses} and $N_{clusters}$ are, respectively, the number of total desired poses and found clusters.

Two pose generation methods where developed.

Sampling Poses Inside a Bounded Volume

This method requires three parameters, both the maximum (r_{max}) and minimum (r_{min}) radius and the center of observation, i.e. the center of the cluster to which the pose will look at. With this information, the pose will have its origin randomly determined inside the defined volume: a sphere with thickness $r_{max} - r_{min}$ and center on the observation point.

With both points, the z versor is calculated pointing to the center of observation. The y versor comes from the cross product between the z versor and a random vector and the x versor is also the cross product between the already defined z and y versors. A detailed explanation can be found on algorithm 3.1.

Algorithm 3.1: Algorithm for correcting orientation of a pose

Input: pose, looking_point
Output: correcly oriented pose
z_versor ←subtract the looking_point to the pose's origin, and normalize it;
rand_vector ←generate a random vector;
y_versor ←normalize(z_versor × rand_vector);
x_versor ←normalize(y_versor × z_versor);
set pose's orientation to the newly defined one by versors x, y and z;

This method, which the result is present on Fig. 3.28, as the disadvantage that it is the user's responsibility to imperatively determine both radius and even after that, due to the imposed orientation, most of the furthest poses are not reachable by the manipulator.

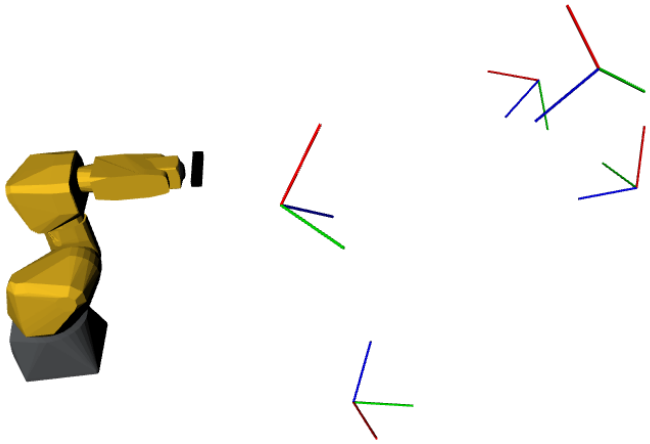


Figure 3.28: Example of five generated poses, with volume bounded pose sampling method, poses looking towards the center of the unknown space (hidden for better visualization). The camera frame axis are blue.

This issue led to the development of a new pose sampling method.

Sampling Poses Bounded by the Robot Reach

The orientation component of the pose was inherited from the previous method. For the position component of the generation, they are randomly sampled within the reach of the manipulator using MoveIt. To prevent poses where the manipulator itself occluded the exploration volume, the position generation is limited to the front hemisphere of the working space.

The orientation of the pose is driven by the cluster it is looking at (see Fig. 3.29.

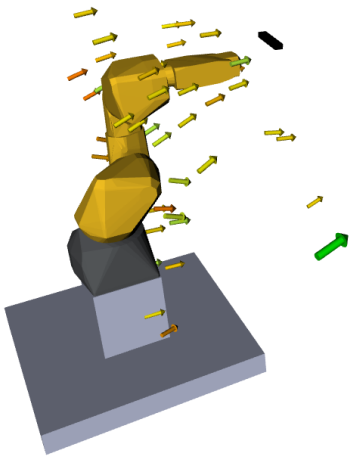


Figure 3.29: Example of fifty generated poses, with robot's reach bounded pose sampling, looking towards the center of the unknown space (hidden for better viewing).

Both these methods give a solution to produce a set of plausible poses. To decide which to visit we need to score them all. The first request to give score to a pose is knowing how much unknown voxels it unveils.

3.12.2 Assessing How Much Unknown Space May Be Discovered by a Pose

The previous sections have described several methodologies that generate poses from which the manipulator observes the scene in question. Thus, the next step is to be able to evaluate each of those poses. The evaluation of these poses is based on the estimated ammount of volume that is (or may be) observed by the robot when positioned on that pose. As noted before, we are simply estimating the volume that may be discovered by a new pose. This is an estimate since it is not possible to be sure if some regions of the yet to be seen volume will be occluded.

To estimate the volume that may be observed, we propose two algorithms: one based on ray casting over all directions of the image pixels, and another based on ray casting driven not by the pixels but rather by the centers of the unknown voxels that lie inside the camera frustum.

Figure 3.30 shows an example the evaluation of a pose where the voxels expected to be known, if the robot is placed on that pose, are colored blue.

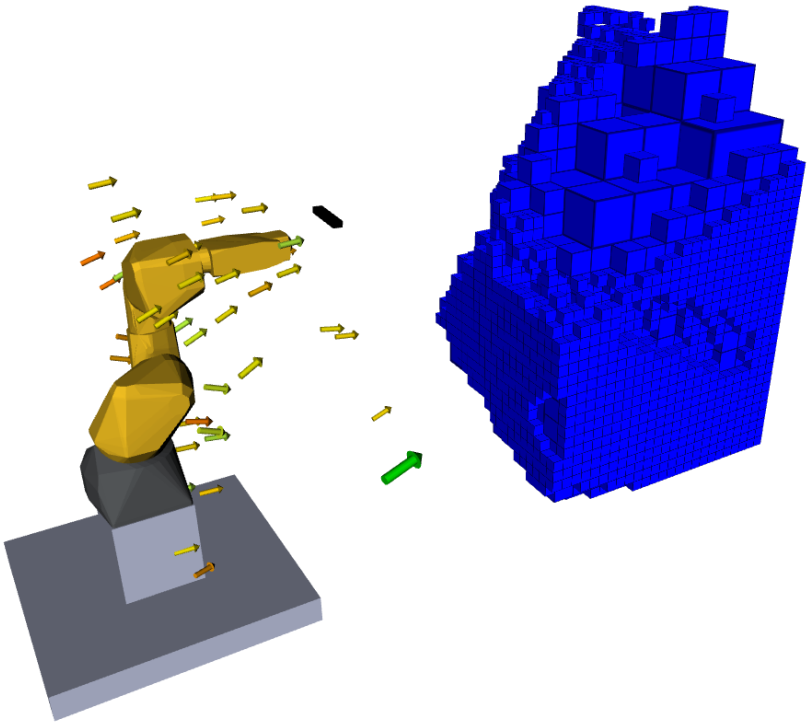


Figure 3.30: Voxels expected to be evaluated (blue) based on the best pose of those sampled, colored as green.

Two methods for retrieving this information are now introduced.

Pixel Based Ray Casting

To evaluate whether or not the unknown voxels will become known, several rays that represent the viewing directions of a pose are casted in 3D space. Then, the goal is to assess, for each of those rays, if they intersect unknown voxels. When this occurs, then the intersected voxel is likely to be observed by that pose.

Given this, the most straightforward option to select the viewing directions is to define one direction (i.e. one casted ray) for each pixel in the camera image. This section describes how this pixel based ray casting was designed and implemented.

Knowing the resolution of the sensor we can compute the set of directions that connect the pose we intend to evaluate to each pixel. This directions are the ones that define the rays to be casted. Casting a ray means getting the identification keys of the voxels that are intersected by a straight line connecting the pose to the end point of the ray, which can be defined by two situations, either a occupied voxel is hit or the ray reaches the outside of the frustum. This ray casting occurs to all previously computed directions.

The set of voxels that are unknown and are intercepted by, at least, one ray are the ones that are likely to be observed by that pose. Algorithm 3.2 explains this procedure.

Algorithm 3.2: Pixel based ray casting

```
Input: pose, near_plane_distance
Output: list of voxels expected to be known
Initialize directions list, Dirs ←{};
foreach col ∈ image do
    foreach row ∈ image do
        direction ←compute direction from pose's origin to pixel image[row,col];
        Dirs = {Dirs, direction};
    end
end
foreach direction ∈ Dirs do
    ray_endpoint ←cast ray from pose's origin, with computed direction;
    ray_keys ←get the keys of all voxels passed through by the casted ray until
     reaching ray_endpoint that are further than near_plane_distance;
    foreach key ∈ ray_keys do
        key_voxel ←get voxel correspondig to that key;
        register the voxel according to its order of passing;
    end
end
```

This method works but in practice it is often unfeasible to perform ray casting for a large number of pixels. Furthermore, several neighbor pixels often cast rays which are very close and intersect mostly the same voxels. This problem is amplified since we aim to evaluate several poses in each exploration iteration.

One option is to subsample the pixels that produce ray directions. This, however, is not a very good solution since it subsamples uniformly over 3D space. For example, areas which potentially contain more undiscovered information (more unknown voxel) could benefit from a higher local resolution of casted rays.

Voxel Based Ray Casting

To solve the problem of subsampling pixels that produce ray directions, this method starts the ray casting on the unknown voxels instead of on the pose. This not only has the potential to give higher resolution to locals with more unknown voxels but, also, significantly decreases the amount of rays to be casted.

If we start the casting from the furthest voxel to the closest (relative to the pose to be evaluated) there is a high chance that the first rays intersect a large amount of voxels. Assuming that, in this evaluation, a voxel only needs to be visited once for its state to be defined, there is no need to cast a ray again for those voxels (that have already been intercepted), eliminating them from the queue, reducing the overall rays to be casted.

After computing the center point of all unknown voxels, we start by defining the set of those that lay within the frustum, since only these have the potential to be discovered. These points, and there corresponding voxels, are sorted by distance to the pose being evaluated. Starting from the furthest voxel, the direction from the pose to its center point is computed. This direction is used to cast a ray that, if there is not any occlusion, will pass through the voxel. All the voxels that are intercepted by this ray will not be visited again, since we already have the confidence that, at least, one ray will hit them. In the case where the ray ends in a occupied voxel it is needed to take in account the voxels that are occluded. So, following the same direction, from this occupied voxel until the end of the frustum, those voxels that would be intercepted by this hypothetical ray will not be visited any time and neither be assumed to be discovered.

Then, for the next furthest voxel that has not been visited yet, the process repeats itself until all the volume is evaluated. Algorithm 3.3 describes this process.

Algorithm 3.3: Voxel based ray casting

```
Input: pose, near_plane_distance
Output: list of voxels expected to be known
get the voxel centers that lay within the camera's frustum;
foreach voxel inside frustum do
    get the distance to camera;
end
sorted_voxels ←sort voxels from furthest to closest to the pose;
foreach voxel ∈ sorted_voxels do
    if voxel is marked to be visited then
        direction ←compute direction from pose's origin to voxel's center;
        ray_endpoint ←cast ray from pose's origin, with computed direction;
        ray_keys ←get the keys of all voxels passed through by the casted ray
         until reaching ray_endpoint;
        foreach key ∈ ray_keys do
            key_voxel ←get voxel correspondig to that key;
            distance ←compute distance from pose's origin to key_voxel;
            if distance > near_plane_distance & key_voxel is unknown then
                register the voxel according to its order of passing;
                set voxel to not be visited again;
            end
            if key_voxel is occupied then
                set all voxels after key_voxel to not be visited;
            end
        end
    end
end
```

Image 3.31 represent the visual outcome of this algorithm, where is visible the casted rays (gray lines), the occupied (red), free (green), unknown (orange) and expected to be known (blues) voxels, as well the camera frustum.

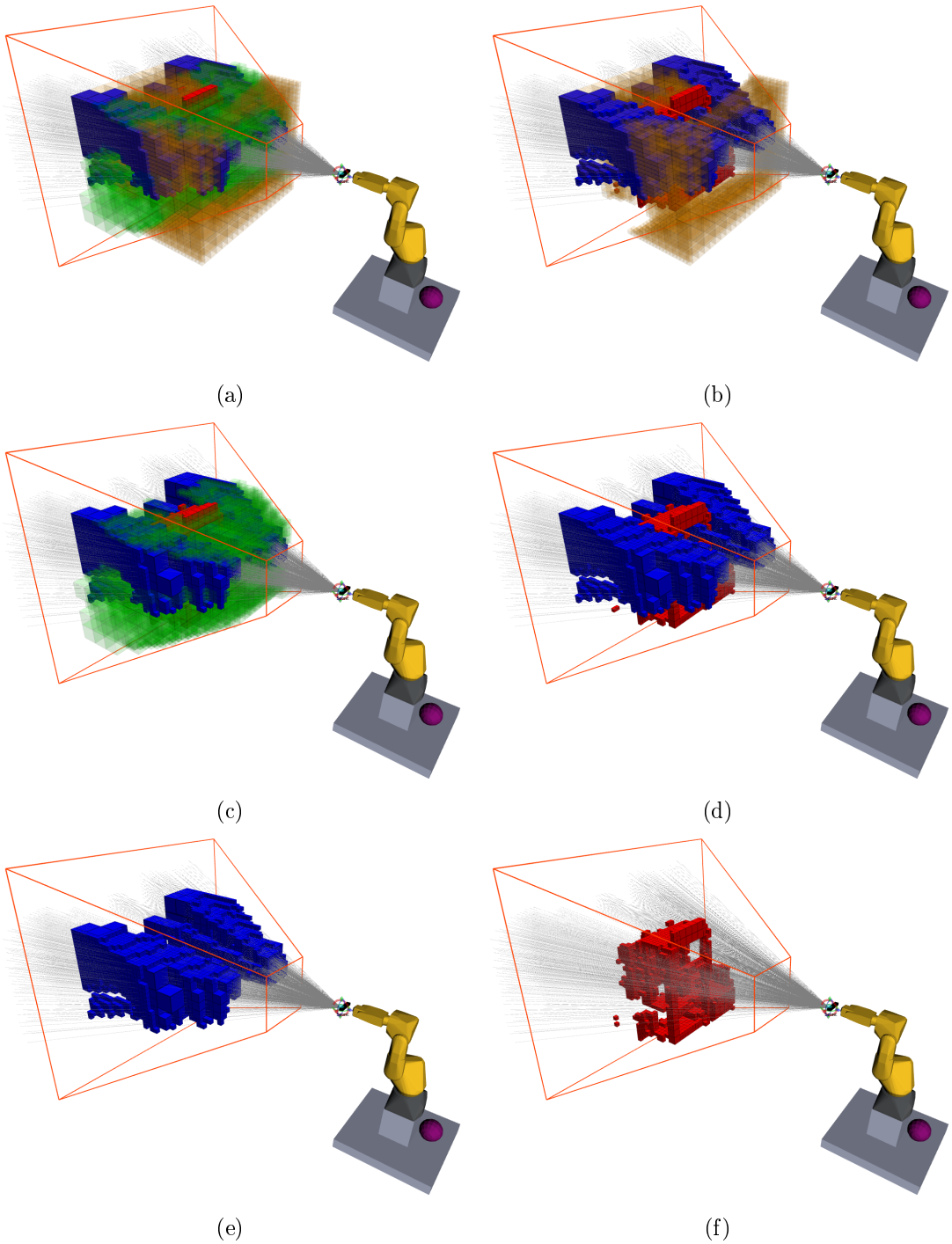


Figure 3.31: Evaluated pose. This Fig. show the beams passing the free space (green) and stopping when they it an occupied voxel (red). The voxels that potentially will be known are marked in blue. In (a) everything is showing. In (b) it is visible the unknown, occupied and expected to be known volumes. (c) is like (b) but the unknown volume was removed and the free added. In (d) only the occupied and expected to be known volumes are showing. In (e) and (f) are only shown, respectively, the expected to be known and occupied volumes.

Going back to the proposed in section 2.5, we want to rank the poses against each other. The metric implemented to do so, i.e., the formulation that gives a score to a pose is based on this assessed unknown volume to be discovered by a given pose.

3.12.3 Pose Scoring

To rank all the poses, a term of comparison has to be set. In this dissertation, when estimating how many voxels a pose will evaluate, we expressed this value as a volume. Then, it becomes natural that the term of comparison should also be a volume.

As refereed in both algorithms 3.2 and 3.3, the order in which the voxels are intersected is useful information. The first expected to be intersected voxel is always guaranteed to be discovered, yet all the following ones will depend if the first one (or any in there front) is actually free or occupied.

Since the quantities of voxels that are the first to be intercepted by a given ray, n_{first}, and that are intercepted by a ray but have an unknown voxel between them and the evaluated pose, $n_{posterior}$, are known, as well as the OctoMap's resolution , a weight w can be set to give less relevance to the voxels which are uncertain to actually by observed,

$$score = \frac{(n_{first} + n_{posterior} \times w) \times resolution^3}{volume_{outer} + volume_{inner} \times w} \tag{3.2}$$

where $volume_{outer}$ is the aggregated volume of all voxels that have at least one face exposed, and the $volume_{inner}$ is the volume of all voxels that have all their faces connected to another unknown voxel. Both volumes are computed before the exploration begins and have the objective of normalizing the pose's revealed volume in a similar way as it is estimated.

As result, Fig. 3.32 demonstrates a scored pose. The more proximate the frustum's color is to dark blue, the higher the pose's score is.

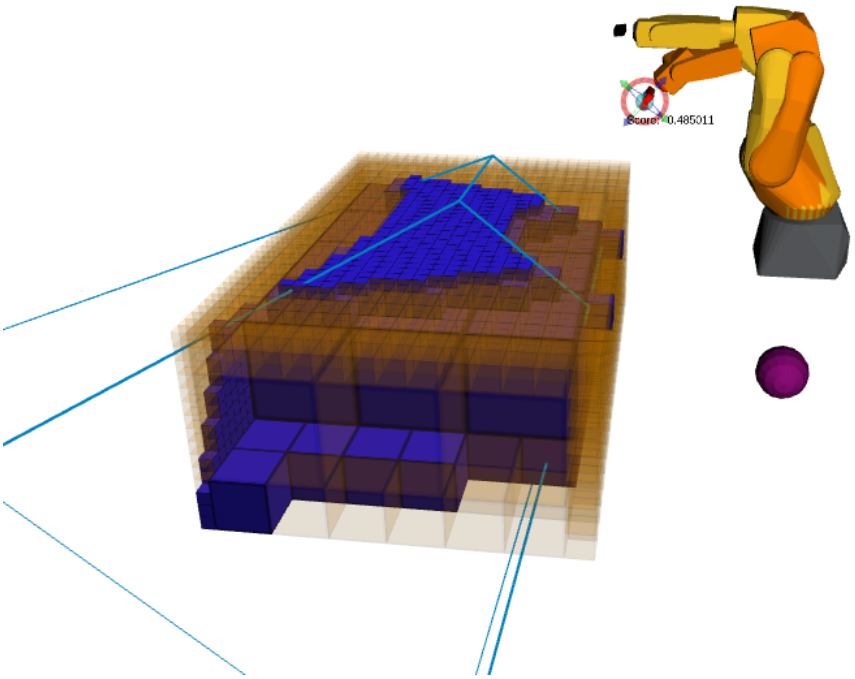


Figure 3.32: Scoring of a robot's new pose. The darker the frustum color, the higher the score.

Scoring a pose is also useful in the sense that provides a stopping criteria for the process. When a set of poses are sampled and the best of their scores does not get above a given threshold (i.e. $\max(score_k) < threshold, k \in \mathbb{N}_{n_{poses}}$, where n_{poses} is the defined maximum number of poses to be sampled), that pose is still visited since it was evaluated but will be the last given that, from this pose onward, the information gain will surely be below the values pretended. This can occur by two factors: either all the unknown volume has actually become known, or the voxels that remain unknown are on unreachable places.

Having gathered the desired, possible, information about the previously unknown scene, the process finishes successfully.

3.13 Summary

The system is now capable of perceiving its exploration goal, i.e., the volume it is requested to explore. This goal is the only part that is not autonomously done, it has to be a human define it. This exploration volume definition implies an automatic outlining of the point cloud spatial filter volume, which will limit the quantity of date fed to the system.

In this exploration volume, the algorithm is able to perceive which voxels it has not gathered enough information from. This information about what it does not know is the starting point for evaluating a pose. This pose can be given by the user or, in an autonomous mode, obtained by sampling a fixed number of plausible poses.

For the poses generated to be plausible, the algorithm clusters the volumes it still unknowns, if they are connected. This allows for the poses to be looking towards different points and, for sure, be gathering some new information. Then, resorting to ray casting, the amount of information given by every single pose is assessed, for ranking to rank them, choosing the best, reachable one, moving towards it, while continuously recording the measures taken.

In the next part, two case studies that assess the reliability and portability of the algorithm to other manipulators are provided. Finally, its intelligence is tested against humans.

Intentionally blank page.

Chapter 4

Results and Discussions

This chapter presents results of the developed system and, when opportune, discuss its limitations, applications and possible improvements in a future continuation.

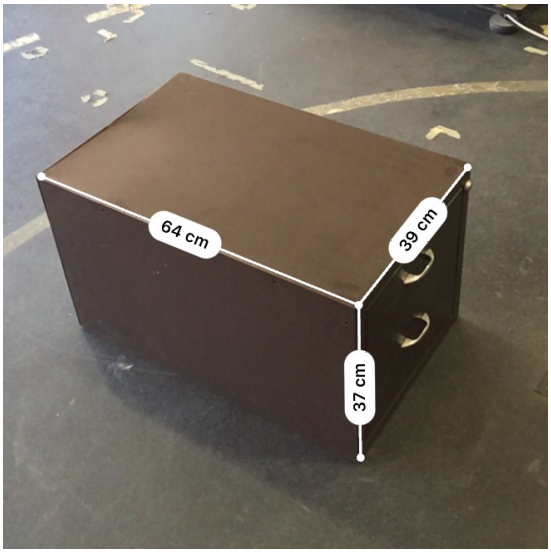
The first topic to discuss is the functionality and autonomous capabilities of the developed system and for that, two case studies will be introduced.

4.1 Case Study 1: Laboratory of Automation and Robotics

On this case study the focus was to evaluate how the system reacts to a couple of challenging environments. The idea was to maintain all the hardware architecture used in the development. The change in hardware will occur on Case Study 2: An Industrial Application) to infer the autonomous exploration capabilities.

4.1.1 Shelf and Cabinet Scenario

This first scenario consisted of a cabinet (Fig. 4.1a) that has a shelf (Fig. 4.1b) behind it. The shelf has some objects on the shelves to generate occlusions, and, on the cabinet, one of the drawers is semi open, and contains an object inside, creating a volume that can not be seen and other that can only be visualized from a narrow set of viewpoints. The complete scenario can be seen on Fig. 4.2.



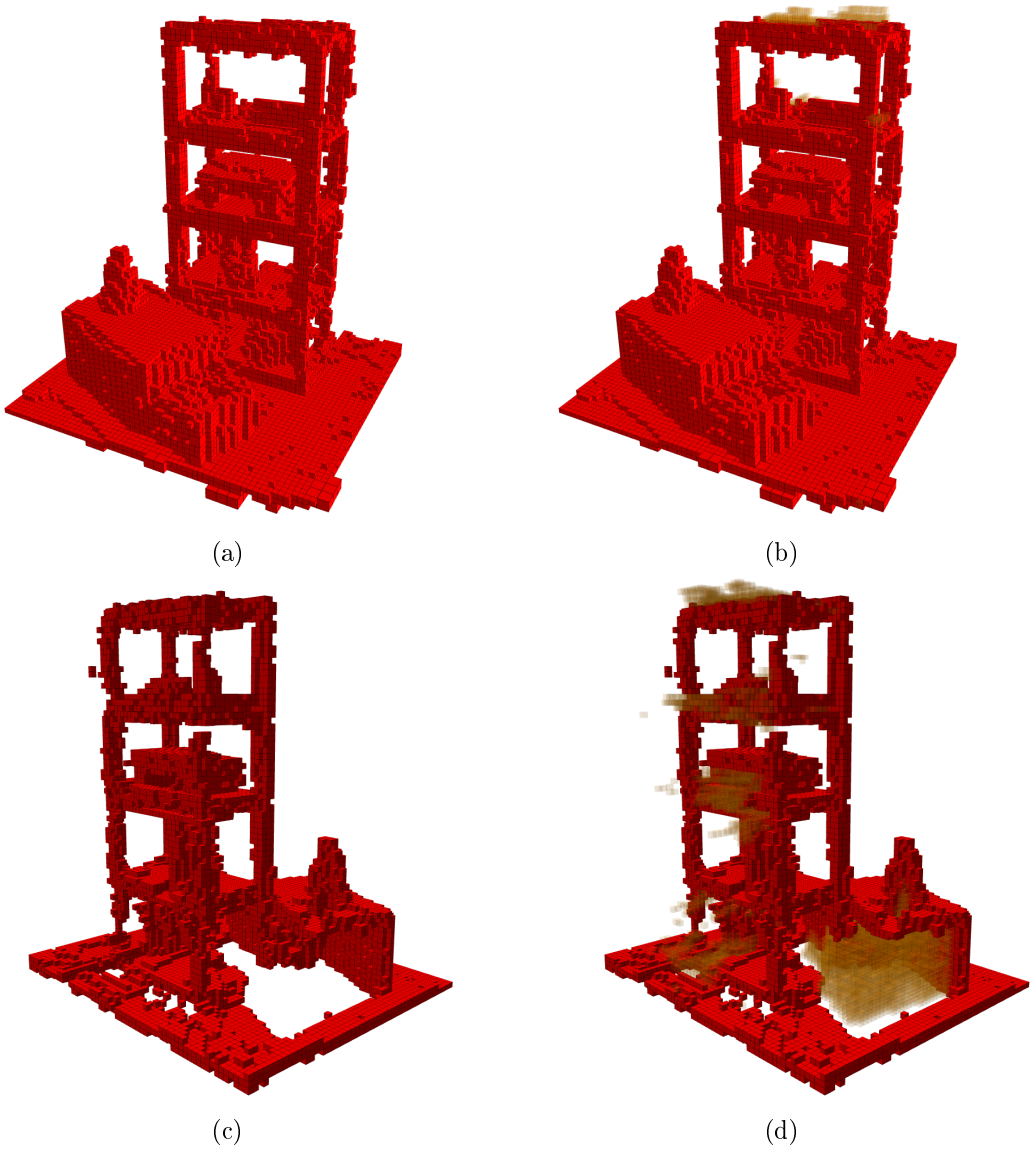
(a)

(b)

Figure 4.1: Measures of the used components of the first scenario. (a) and (b) show, respectively, the measures of the used cabinet and shelf.

Figure 4.2: Complete shelf and cabinet scenario.

From the reconstruction of this scene resulted Fig. 4.3, in which are visible the front and back views. On Fig. 4.3d we can notice the volume, inside the cabinet, that will remain unknown. This was expected, since it was a completely surrounded volume, with no visualization window.

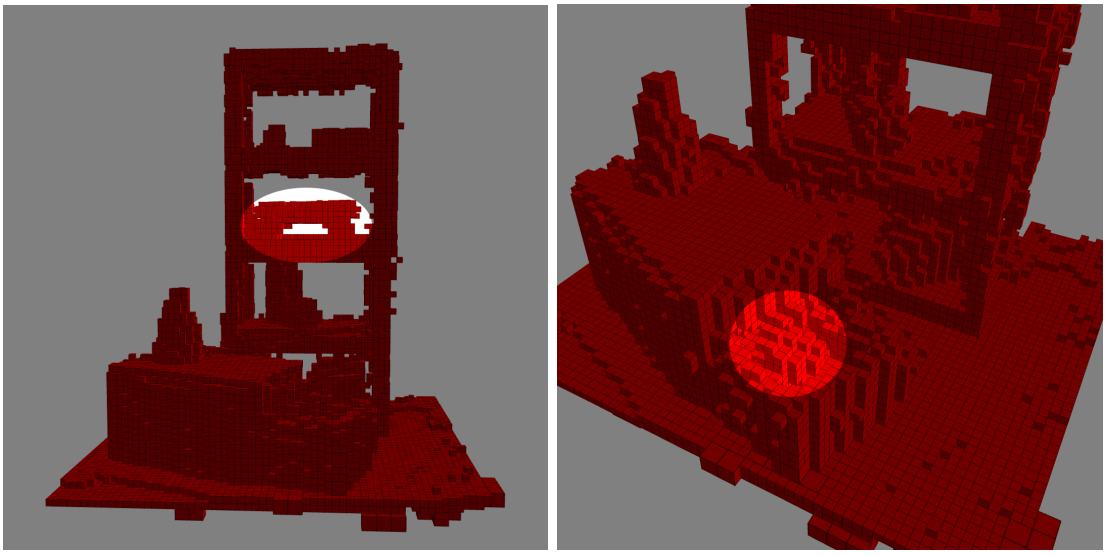


(a) (b)

(c) (d)

Figure 4.3: Complete reconstruction from the shelf and cabinet scenario. Voxel size is 25 mm. On top is a front view and on the bottom is a back view. On the left are only the occupied voxels and on the right are also the unknown voxels. All images were taken after the reconstruction process ended.

For this reconstruction was defined that the finest resolution was 25 mm since it gave the most details without a big compromise in performance. This resolution allows the capture of finer details, as witnessed in Fig. 4.4. On Fig. 4.4a the empty space is noticeable under the stool top (in the middle shelf), while Fig. 4.4b provides a view point where the object inside the drawer is perceivable.



(a) (b)

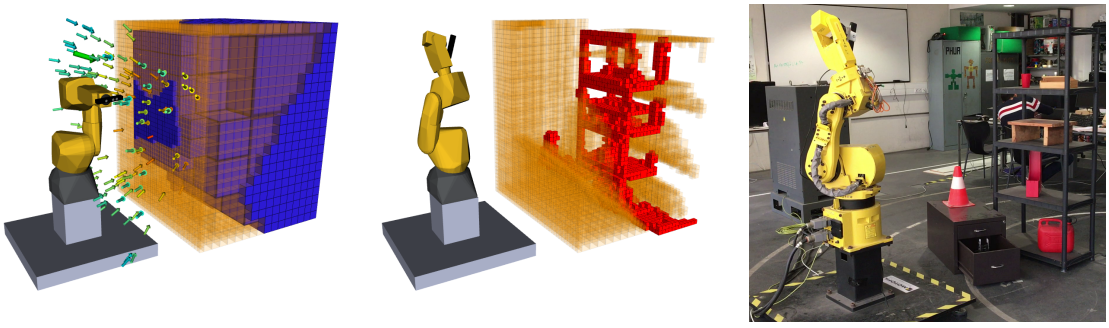
Figure 4.4: Details of the reconstruction. On (a) it is visible the hole of the stool positioned on the middle shelf. On (b) the reconstruction of the object inside the drawer is noticeable.

Figure 4.5 describes the exploration process for this scenario, taking six iterations and sampling 150 poses for each, with the finest map resolution defined at 40 mm. The exploration volume is 3.238 m^3 and the reconstruction ended when the Next Best View (NBV) had a score less than 0.1%.

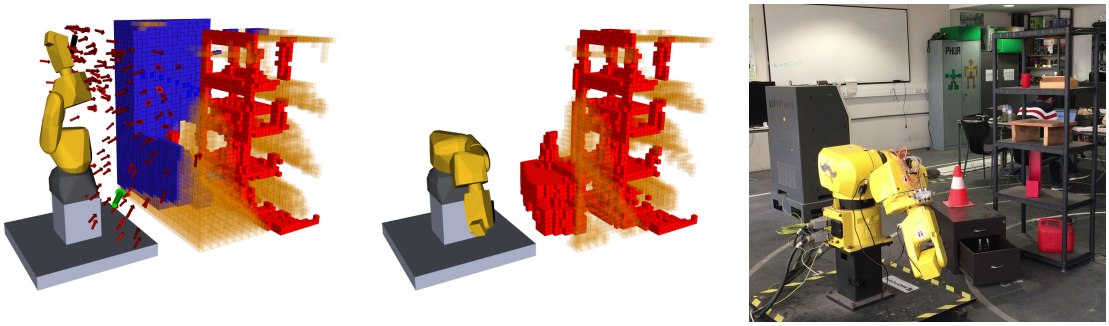
On Fig. 4.5b, is discernible the not inclusion of some voxels in the expected volume to be known, as result of the occlusion created by the traffic cone. Yet, after the pose was reached, we can see that the refereed voxels have in fact been evaluated (in this case, as free) in conjunction with some other in front of the cabinet. This happens as consequence of the path taken by the camera. While moving, the camera is continuously sensing what is inside its Field of View (FOV), sometimes providing information where it was not expected to be collected, depending on the path traveled. This factor was not taken in account during evaluation of poses because it would request a great amount of computational power to evaluate all steps of the path (these steps can add up, surpassing the hundreds).

Iteration three (Fig. 4.5c) demonstrates a characteristic movement of the robot, going from right to left in order to access the state of the voxels positioned where the camera has not looked up until this moment. This NBV was oriented to look towards the bigger cluster in the front right of the exploration volume. Nevertheless, some voxels belonging to different clusters are also expected to be visible and within the FOV, which increases the score of that pose. Because some where behind a solid obstacle (yet unmapped) not all of them where actually observed.

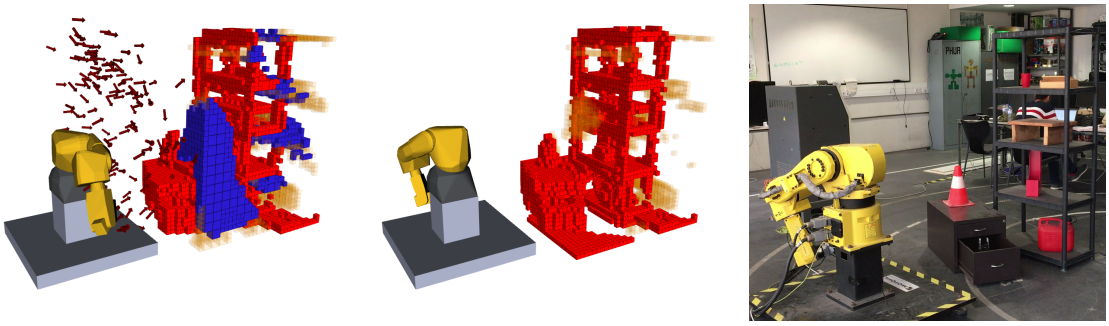
In the next few iterations (Figs. 4.5d, 4.5e and 4.5f) the robot bounces right and left gaining information about a decreasingly unknown volume, until the stop criteria is met. In the end of the process is visible the almost absence of unknown volumes (Fig. 4.5g), which has been evaluated to be free or occupied.



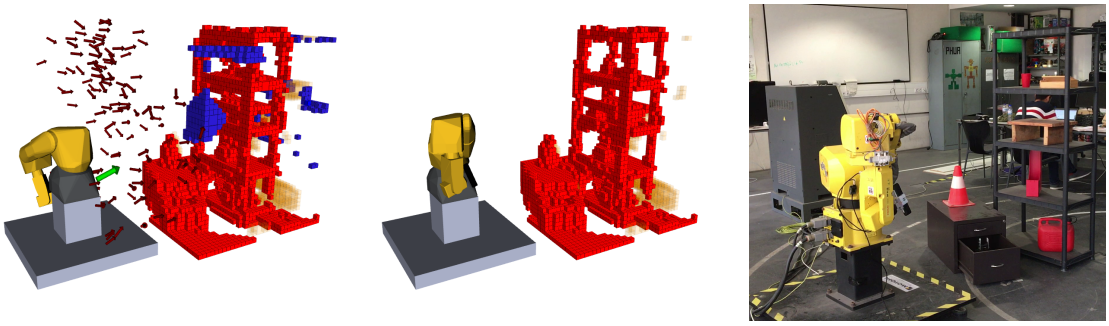
(a)



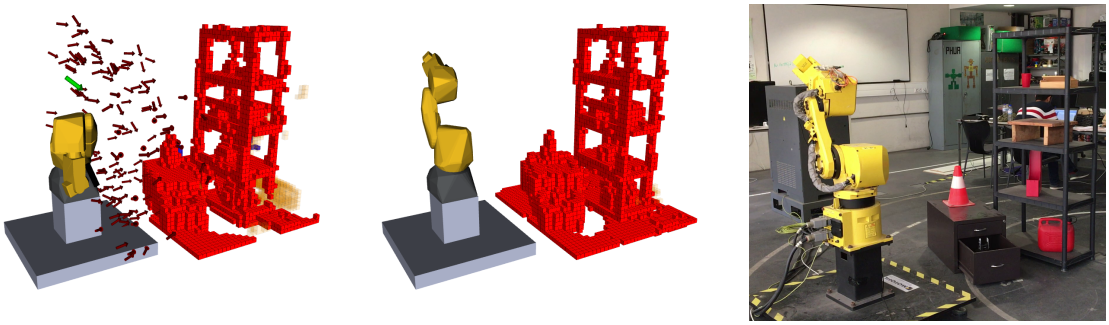
(b)



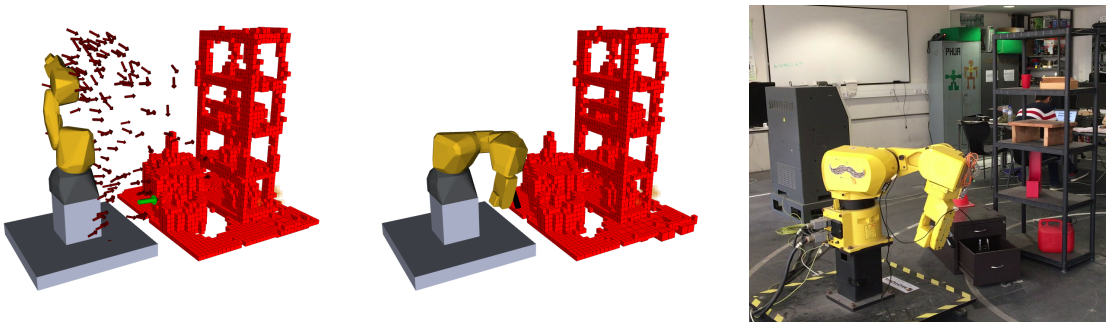
(c)



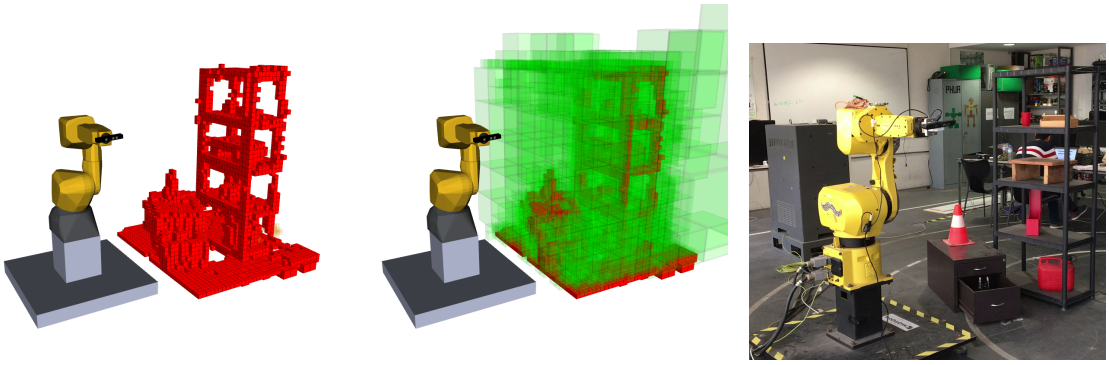
(d)



(e)



(f)



(g)

Figure 4.5: Sequence of iterations for the complete reconstruction of the shelf and cabinet scene. On the left are all the possible poses (the more blue the better score, and the more red the worst), the chosen NBV (in green and bigger), the unknown voxels in orange and the voxels expected to be known on blue. In the middle is what the pose actually improves the model, red voxels are occupied, green represents the free space. On the right is the actual pose of the robot.

The shelf and cabinet scenario provided a challenging reconstruction because of its high complexity, having multiple objects, intricate spaces and volumes impossible to be seen, requesting from the robot several demanding poses to achieve its goal. Despite this, it took an average of eight poses to fulfill the task. The goal for this section was to provide a more qualitative evaluation and demonstrate how the process unfolds, with the quantitative results being discussed on section 4.1.3. This philosophy will be maintained on this next section, where a new scenario is presented.

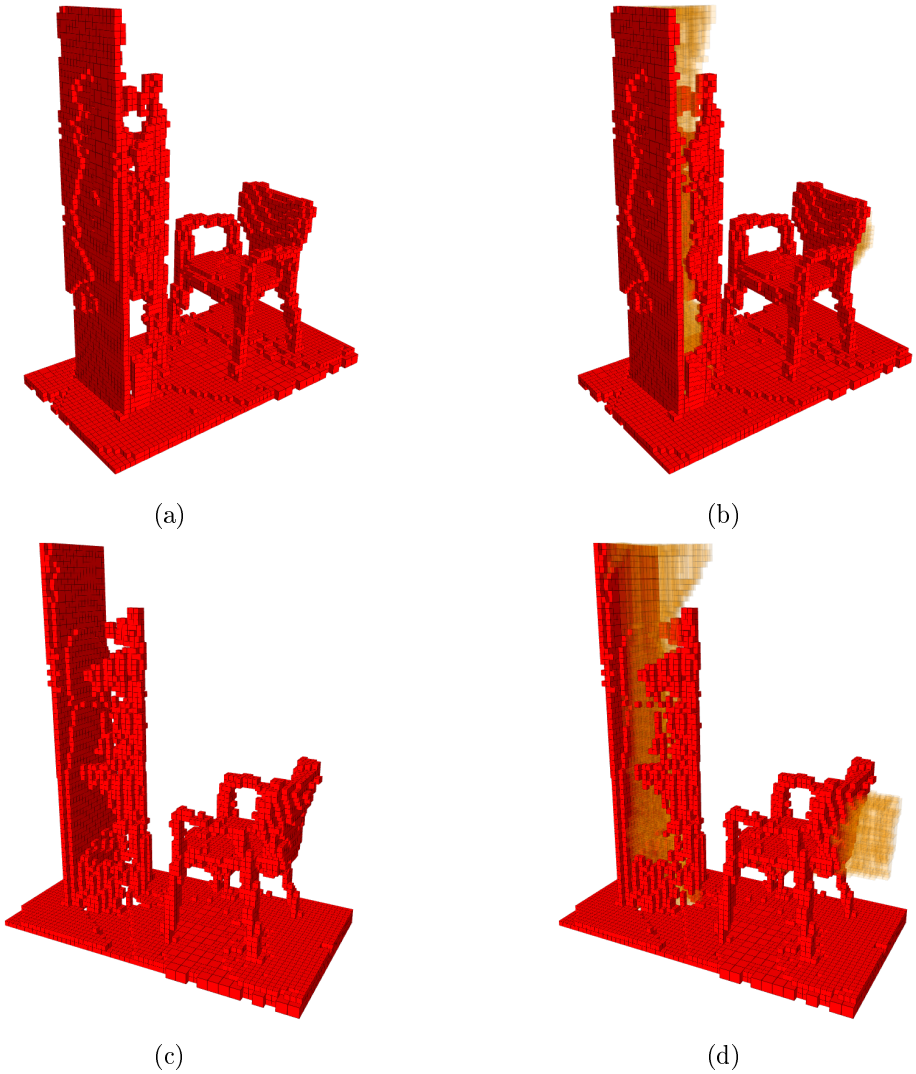
4.1.2 Occluded Chair Scenario

In the previous scenario we were able to perceive that the developed autonomous system is capable of mapping a complex environment with intricate spaces, but it lacked a big occlusion that would highlight the pose evaluation procedure, testing its real aptitude to deal with these kind of challenges. Going even further, the system must be able to react to an occlusion that is expected to happen and move around it, visiting poses that are expected to return the biggest possible information, without interacting with the scene, i.e. without colliding with it.

This new scenario is presented in Fig. 4.6, where the chair is visibly occluded by a tall obstacle with a triangular base. Being this tall forces the robot to look from either side, disfavoring central positions, while making it impossible – given the reach of the manipulator – to look inside of said obstacle by its only entry, at the top. The obstacle is also close enough for the robot to collide with it, if no collision checking would be performed.

Figure 4.6: Complete occluded chair scenario.

Reconstructing this scenario resulted on the model of Fig. 4.7, where not only are visible the front and back views, but the occupied and unknown voxels as well, at the end of the process. Both the chair and the barrier (created by the obstacle) stand out in the model. Although the barrier does not seem completely planar (consequence of it being slightly rotated in relation to the robot's base link), it is clear that it presents an obstacle when trying to perceive the chair.



(a) (b)

(c) (d)

Figure 4.7: Result of the reconstruction of the occluded chair scenario. Voxel size is 25 mm. On top is a front view and on the bottom is a back view. On the left are only the occupied voxels, on the right are also the unknown voxels after the reconstruction process ended.

Two big, unknown, volumes were not sensed in this exploration: behind the chair and inside the obstacle. The unknown cluster in the rear of the chair exists solely due to the robot's inability to reach a pose in which the camera can perceive that volume. Similarly, as already referred, the set of poses able to unveil the cluster inside the barrier would be in a very high position looking down. But even these would not grant a complete

evaluation of the said volume, since the obstacle is taller than the FOV of the camera. Therefore, to completely unveil this volume, the robot had to reach a high position and then, probably, go inside it in a second one. Unfortunately these poses and movements are out of the manipulator's reach.

Another aspect is the poor reconstruction of the back portion of the obstacle, once again explained by the inclination of the plates. As represented in Fig. 4.8 – which shows a detailed view of the inside of the obstacle –, some voxels, in the same vertical line, are occupied and others free, when was expected for all of them to be in the same state. The most plausible explanation lays in the walls orientation. Given the reachability of the manipulator, most of the rays intersecting these voxels would be almost parallel to the barrier, passing through them without actually hitting the barrier, wrongly setting them as free. Yet, in another pose, possibly with a higher rotation, these rays actually bounce on the barrier, setting those voxels as occupied. This uncertainty would explain the high entropy of the reconstruction of those two back barriers.

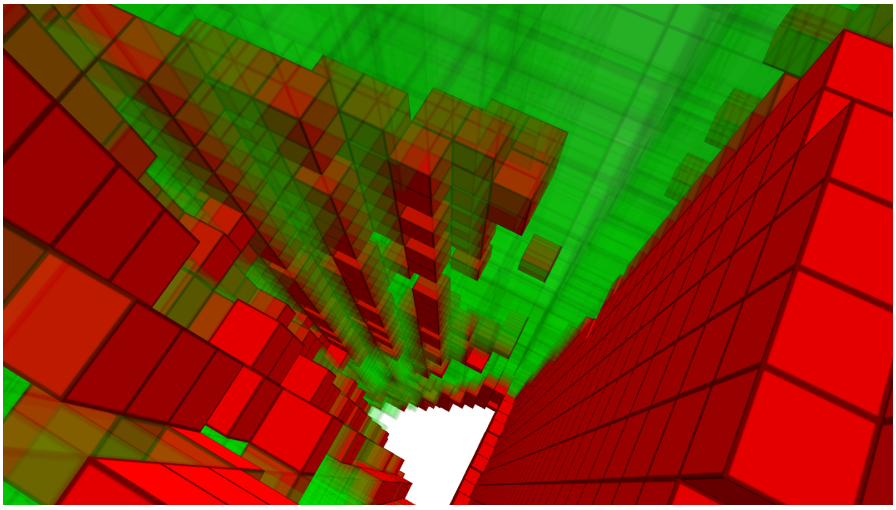
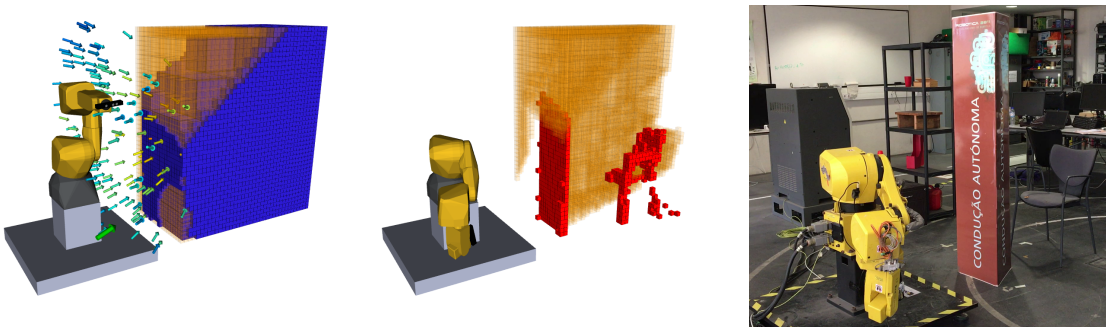
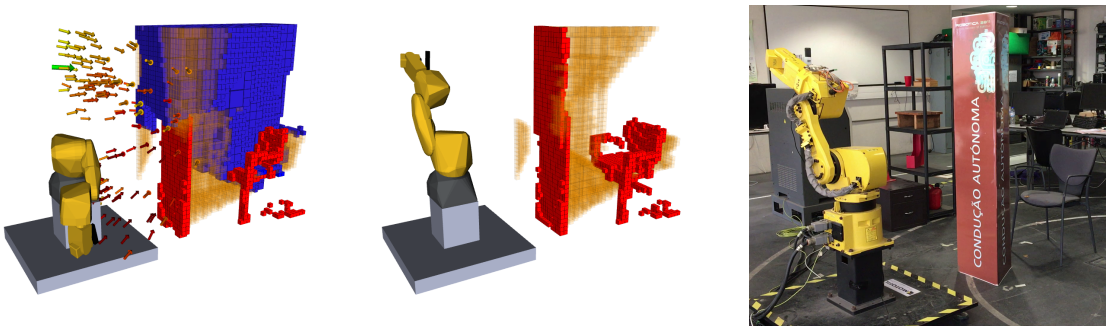


Figure 4.8: Detailed view inside the obstacle of the occluded chair scenario. Red voxels represent occupied space, green represent free space. Unknown voxels hidden for better visualization.

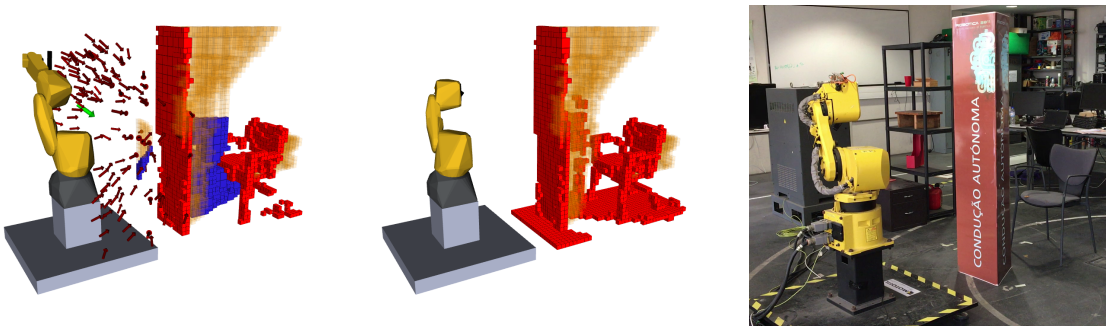
As discussed when analyzing the reconstruction of the shelf and cabinet scenario, the first NBV does not contemplate information about occlusions. By chance, in this first iteration (Fig, 4.9a), only the lower portion of the set is evaluated, requesting a pose capable of sensing the higher portion, which happens in Fig. 4.9b. After the second exploration iteration, the barrier as already been mapped, allowing for full occlusion avoidance poses, exactly as happens in the third iteration, where the robot, almost all stretched, looks from left to right and up to down, to reveal the remaining bottom portion of the exploration volume. This leaves a big cluster in the top of the volume that will force the robot to look into it in the fourth iteration, to then gain some more, predicated, details of the chair (see Fig. 4.9e). Finally, like in the previous scenario, there is a last iteration where a small amount of unknown voxels are measured in order to fulfill the end criteria.


CONDUÇÃO AUTÓNOMA

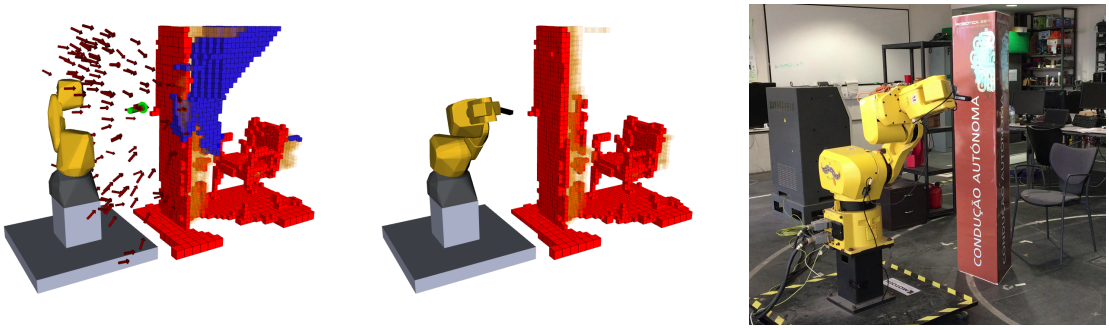
(a)


CONDUÇÃO AUTÓNOMA

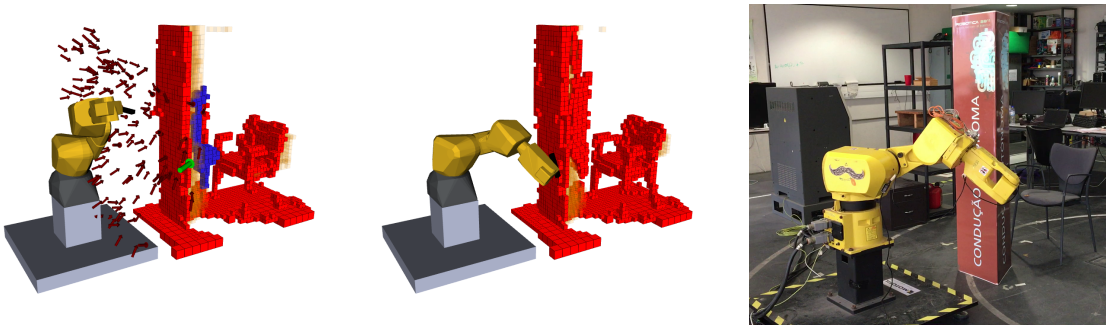
(b)


CONDUÇÃO AUTÓNOMA

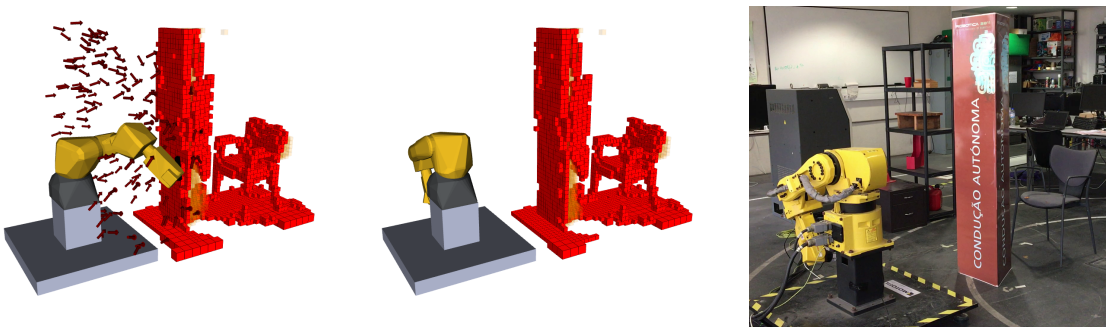
(c)


CONDUÇÃO AUTÓNOMA

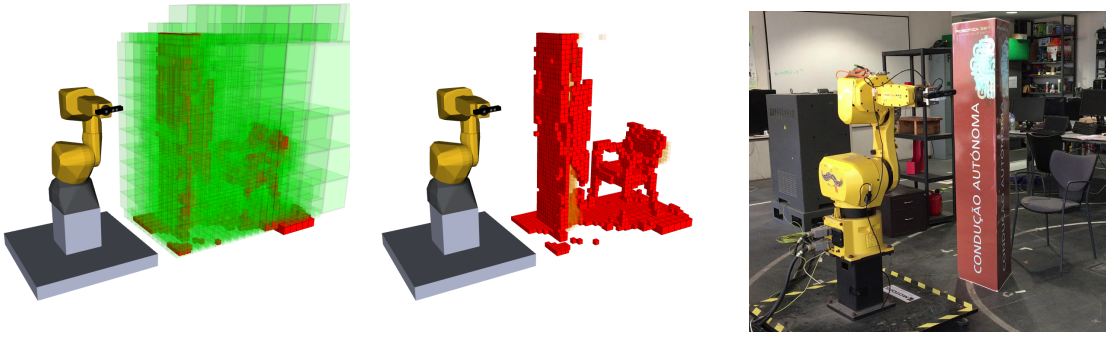
(d)



(e)



(f)

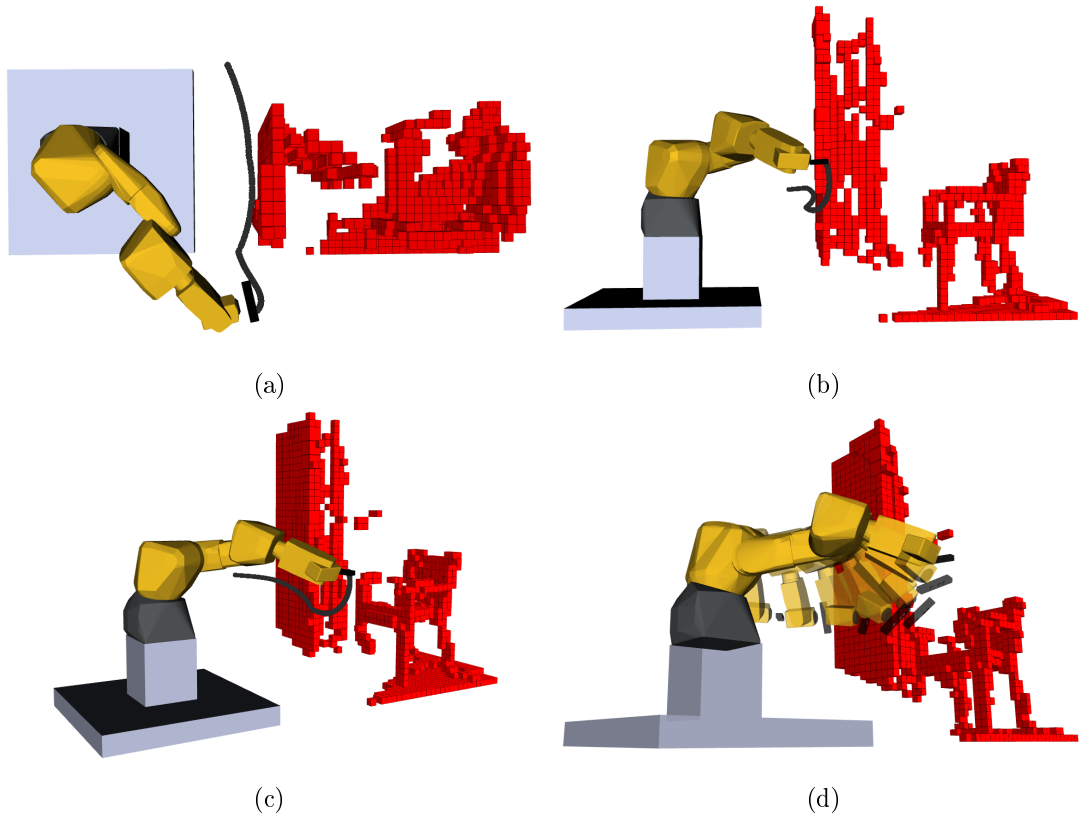


(g)

Figure 4.9: Sequence of iteration for the complete reconstruction of the occluded chair scenario. On the left are all the possible poses (the more blue the better score, and the more red the worst), the chosen NBV (in green and bigger), the unknown voxels in orange and the voxels expected to be known on blue. In the middle is what the pose actually improves the model, red voxels are occupied, green represents the free space. On the right is the actual pose of the robot.

One of the goals when testing with this different scenario was to evaluate how the robot reacted, and planned, to a object within its reach. Figure 4.10 illustrates an example case where this happens. After having reconstructed part of the scene, the robot selected a pose that requested it to move across the scene but, if taken a straight path, would collide with an object it already knows the existence of. To avoid it, the camera is rotated to a vertical configuration (see Fig. 4.10d) curving down and inwards in the process, distancing itself from the barrier. When reached a position where a collision was not expected to occur anymore, an upward path, while rotating to the requested configuration, was taken to achieve the goal pose.

In a similar way as happened for occlusions, this behavior tends to react better to the obstacles as more and more knowledge about the environment is gathered. It also gives the insurance of a completely autonomous exploration by the robot. The ability to generate a model of a totally new environment, without changing it with a collision is there, achieving the goal of this dissertation. Accessing if it is advantageous to use in replacement of a human controlling the manipulator is discussed further ahead in section 4.3.



(a) (b) (c) (d)

Figure 4.10: Path taken by the depth optical frame in order to reach the goal pose, without colliding with the space already known to be occupied. (a) is the top view, (b) is the side view, (c) is a perspective view and (d) is the trail representing several robot configurations during that path.

4.1.3 Experimental Analysis of the Autonomous Exploration Performance

In this section a set of quantitative results are provided. These results compare the technique adaptation of the algorithm to both scenarios already described.

The summary of the physical characteristics and performance of the algorithm are highlighted on Table 4.1.

For the results that follow it was requested to the system to reconstruct both prior scenarios – six times each –, with a stopping criteria defined at 0.1% (accordingly to section 3.12.3) and 40 mm OctoMap resolution. To accomplish this, 150 poses where sampled by exploration iteration. Table 4.1 additionally summarizes the statistical information for each scenario, either the requested number of iterations and also the agglomerated distance between the visited poses.

Table 4.1: Statistical results of the explorations in Case Study 1, shelf and cabinet and occluded chair scenarios.

Scenario	Exploration Volume [m³]	Number of Iterations		Total Distance [m]	
		Mean	Standard Deviation	Mean	Standard Deviation
Shelf and Cabinet	3.531	8.00	1.789	8.525	1.084
Occluded Chair	2.140	5.17	0.408	6.365	1.089

It is evident that when the exploration volume diminishes, less iterations, and therefore poses, are required to fully reconstruct the environment. This leads to a shorter traveled distance by the camera's depth optical frame, but even so, the dispersion of the traveled distance between poses remains approximately the same. This means that the algorithm diversifies its poses no matter the volume to explore, which makes sense considering that this behavior tends to maximize the gain of information by consecutive iterations.

Through measuring both volumes a NBV is expected to reveal and the amount that is actually revealed we can analyze not only if the trend expected is confirmed, but also the precision and accuracy of the computed values.

The first four presented plots (Figs. 4.11, 4.12, 4.13 and 4.14) are all in terms of absolute values, that is, in each iteration we break down both amounts of each volume.

Examining both Figs. 4.11 and 4.12 it is clear a discrepancy in the first iteration (in the order of half cubic meter), favoring a higher expected volume in comparison to those actually seen. This was predictable, as already discussed, because of the nonexistence of occlusions information before the first pose is reached. After the initial measure, occlusions star becoming more predictable, hence, the discrepancy comes down to just a couple cubic decimeters, coming to less than that in the following NBVs.

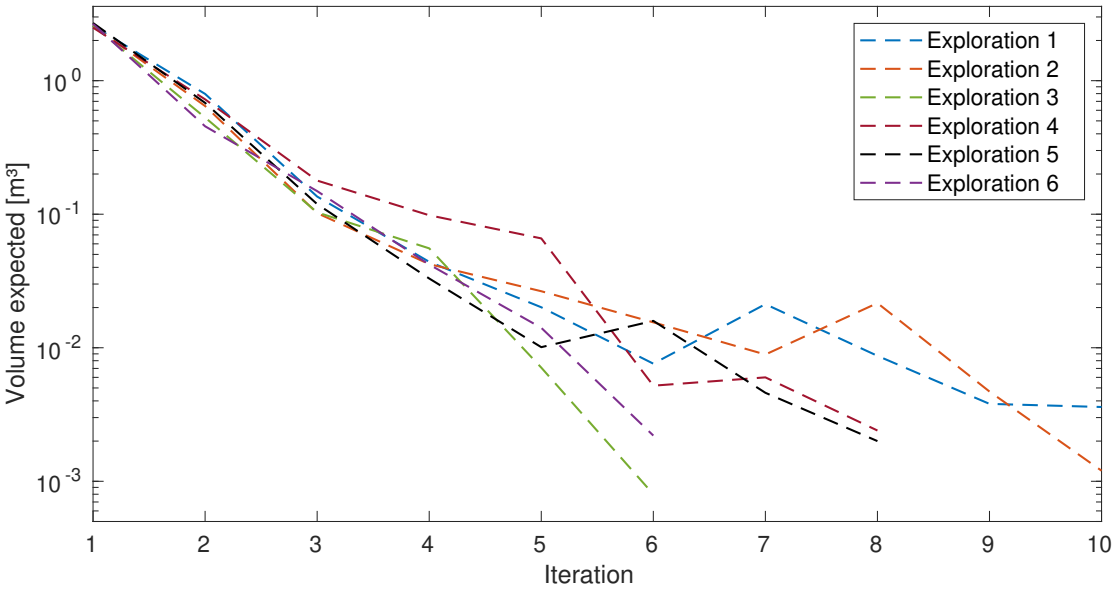


Figure 4.11: Volume expected to be unveiled over the exploration (Case Study 1, shelf and cabinet scenario).

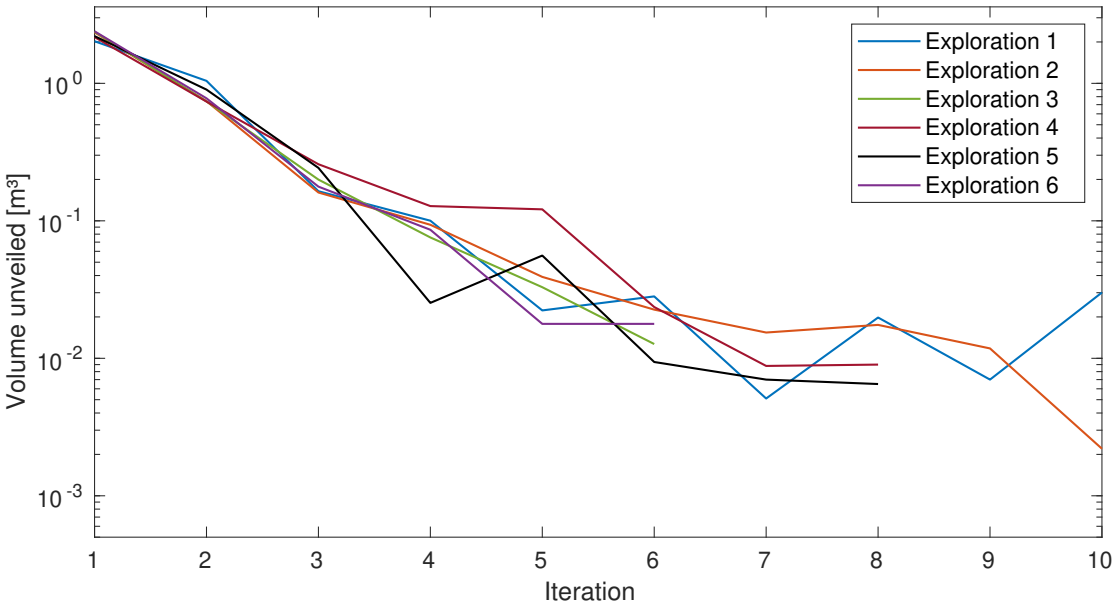


Figure 4.12: Volume unveiled over the exploration (Case Study 1, shelf and cabinet scenario).

In the case of the occluded chair scenario, the revealed volumes in the first iteration are wildly spread (Fig. 4.14) and considerably lower than expected (Fig. 4.13). This is a result of the perturbation in the environment that is the obstacle, promoting big occlusions. The first view always expects to expose large portions of the exploration volume, but then it finds the opaque obstacle that does not allow for further measurements behind it. This contrast is what causes the discrepancy reported by both plots.

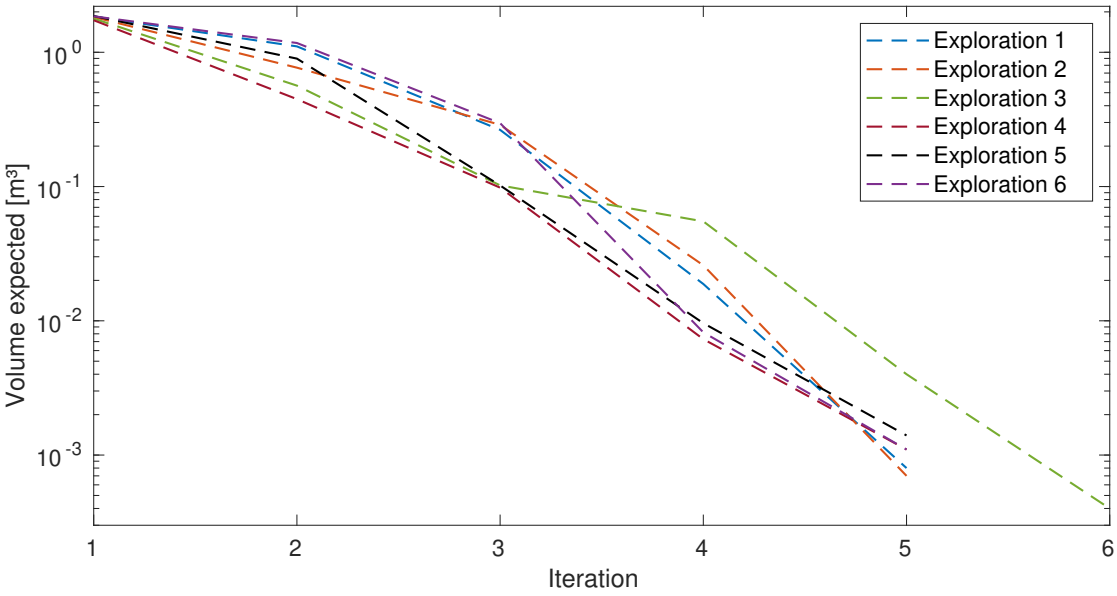


Figure 4.13: Volume expected to be unveiled over the exploration (Case Study 1, occluded chair scenario).

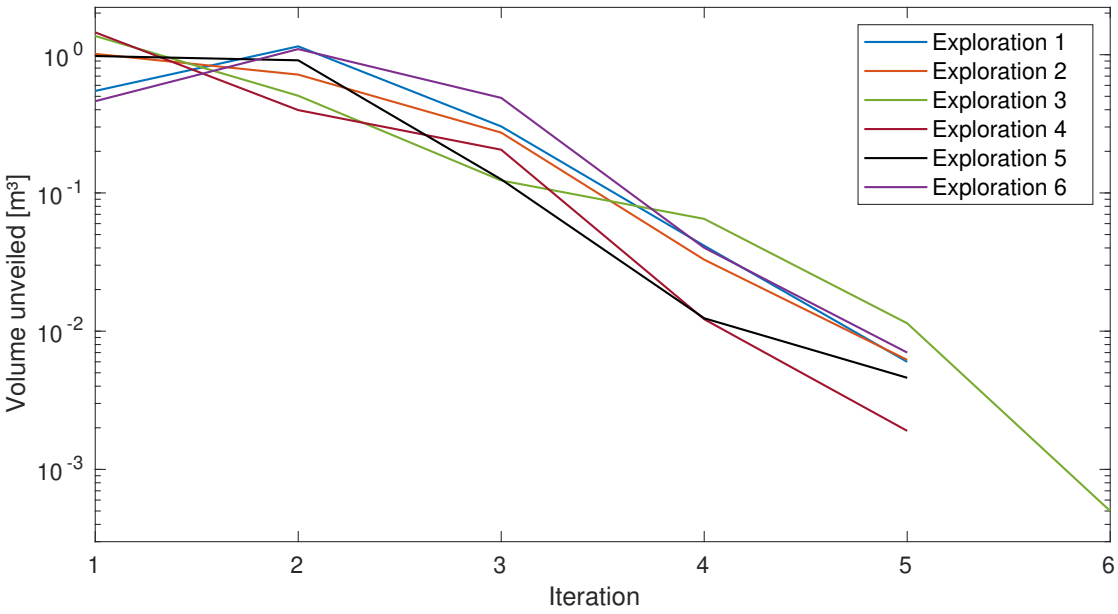


Figure 4.14: Volume unveiled over the exploration (case Study 1, occluded chair scenario).

Nevertheless, this perturbation is then corrected. Having information about the obstacle, the pose evaluation procedure is able to take it into account, providing values closer to reality, proving the adaptability for different scenarios.

Notice that the above plots are all monotonously decreasing. This is because the amount of knowledge about a scene, assuming it does not change, can only stay the same or increase. Given that the camera always moves to a pose where it is expected to unveil some volume and, on the path to reach it, is also evaluating the scene, it never stays the same and always increases. This statement is no longer true if we are studying how is the evolution of the volume uncovered by the NBV relative to the total amount of yet unknown – at that point in time – volume present in the scene.

Equally to what happens when analyzing the absolute values, the first iteration always gives less information than expected. However, this is the only iteration in which this happens, as posterior poses tend to reveal a bigger fraction of the unknown space than predicted. Conclusions are drawn by facing together plots from Figs. 4.15 and 4.16. Taking in consideration that, when moving, the camera is still gathering information about the scene, setting a known state to voxels along the path, explains why this happens, since the evaluation algorithm does not account for the information gain along the trajectory.

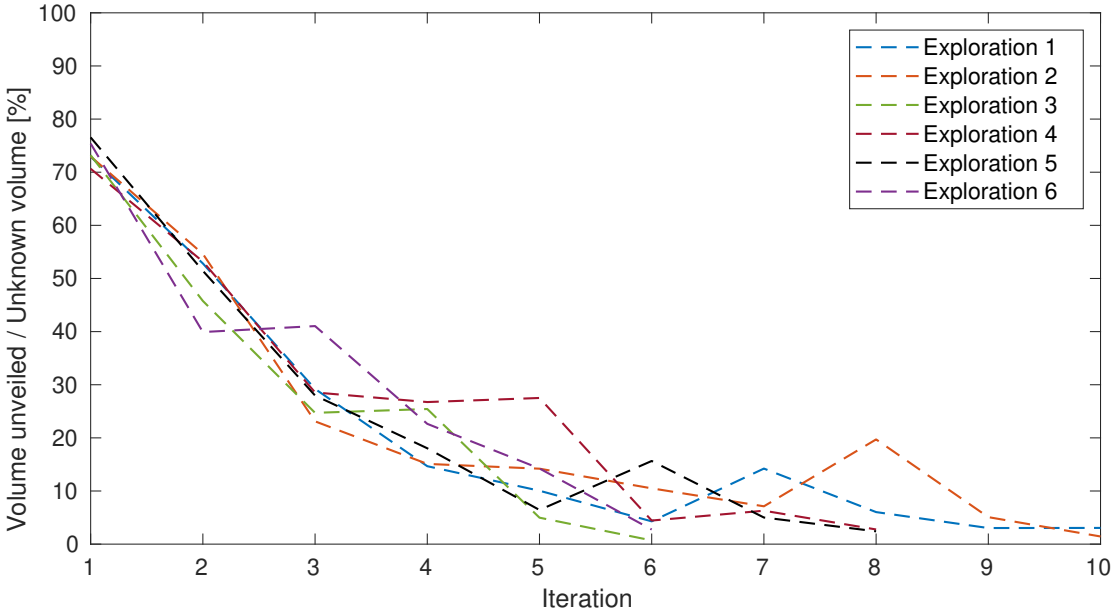


Figure 4.15: Fraction of the volume expected to be unveiled relative to the existing unknown volume, over the exploration (Case Study 1, shelf and cabinet scenario).

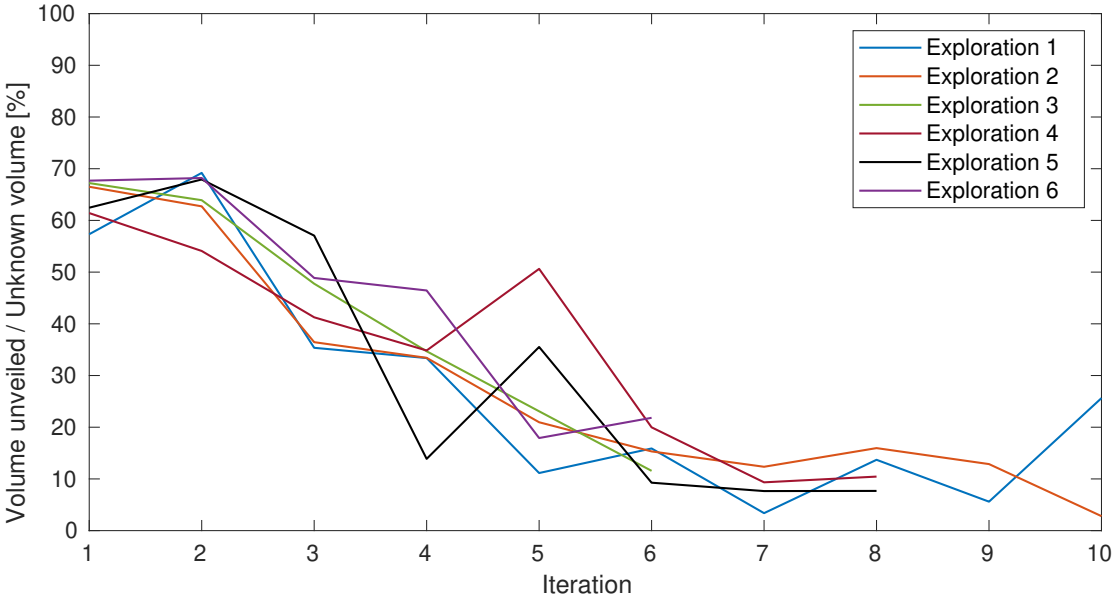


Figure 4.16: Fraction of the volume unveiled relative to the existing unknown volume, over the exploration (Case Study 1, shelf and cabinet scenario).

Yet, these ratio tends to get lower the closer the exploration is to finish because, although there is less volume to unveil, that volume is also more scattered, being harder for a single pose to view all of it.

Supporting the stated about the discrepant values in all first iterations, we can see that the objective of favoring big occlusions in the occluded chair scenario was achieved. For this scene, all six explorations predicted, for their first NBV, the reconstruction of 80% to 90% (see Fig. 4.17), but none even surpassed the 70%, with the lowest barely unveiling 20% of the exploration volume (as in Fig. 4.18). The discrepancy reached its peak on exploration 6, achieving a difference of over 65%. The dispersion of the measured values (Fig. 4.18) is also considerable, comparing to the previous scenario (Fig. 4.16), which was expected, given the complete randomness of the FOV, corresponding to the NBV pose, being mostly blocked or not by the obstacle.

Taking a look at explorations 3 and 6, Fig. 4.17 demonstrates that the difference between their fractional volumes, in the third iteration, is less than really was sensed (see Fig. 4.18). The evident increase of volume revealed from what was expected, for the third iteration on exploration 6 suggests that, when comparing with exploration 3, the path taken was fairly rich in terms of unveiling unknown space.

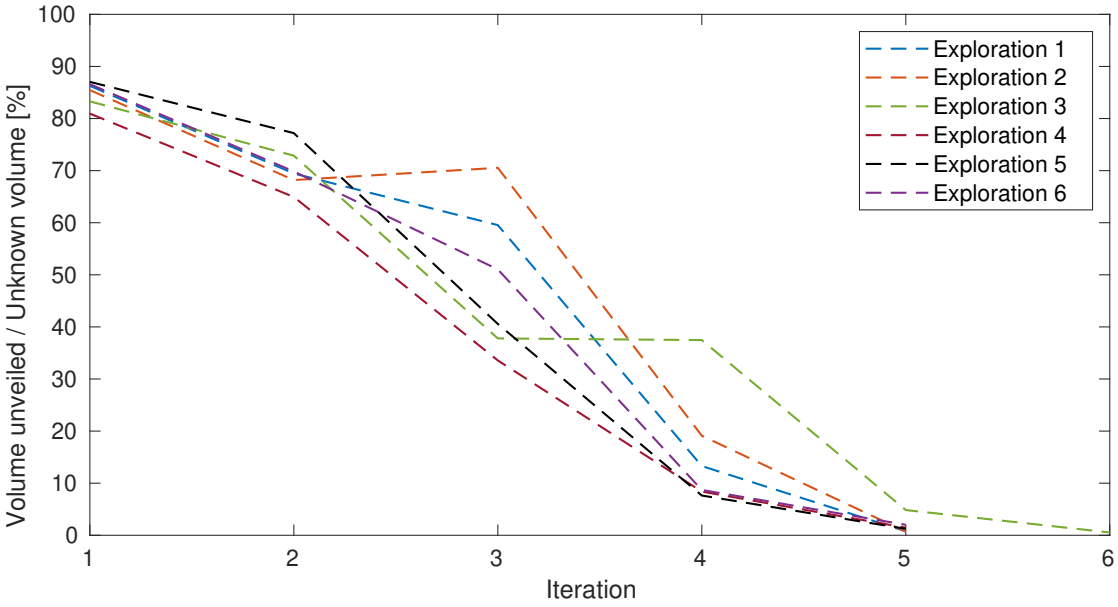


Figure 4.17: Fraction of the volume expected to be unveiled relative to existing unknown volume, over the exploration (Case Study 1, occluded chair scenario).

Given that the path planning computation is not controlled by the autonomous exploration algorithm (and may be different for the same two poses due to the random nature of RRT-Connect), the path itself can or can not be advantageous for an iteration. In the case of exploration 3 it ended up being disadvantageous to the point it required one extra iteration.

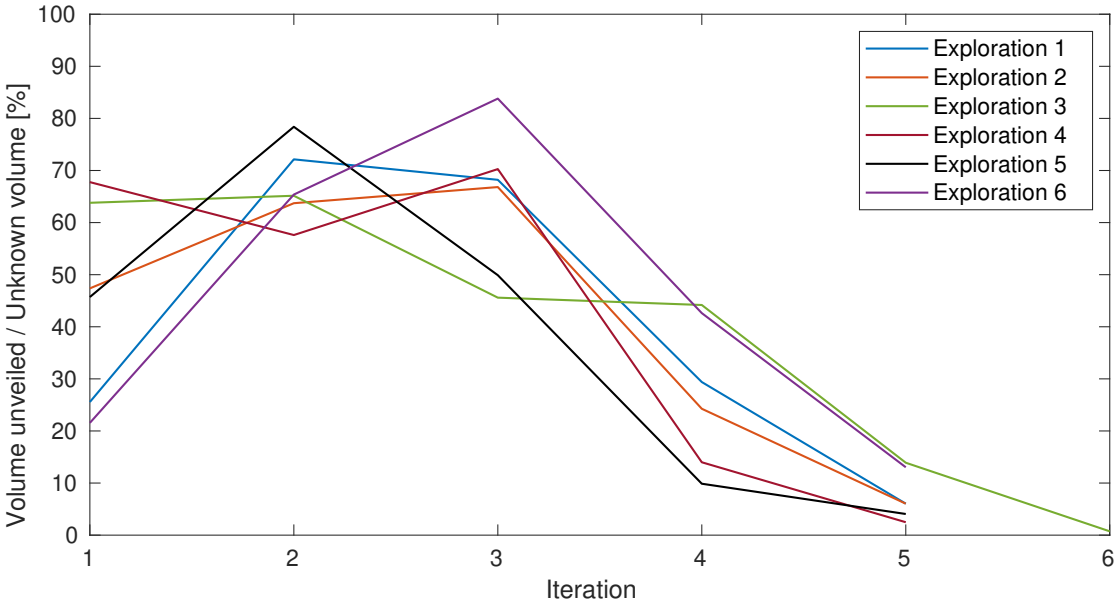


Figure 4.18: Fraction of the volume unveiled relative to the existing unknown volume, over the exploration (Case Study 1, occluded chair scenario).

With that said, the average variation between the fraction that is expected and the one that is actually measured by the sensor (on Fig. 4.19) is practically null just after the first iteration, coming to positive ground (almost 20%) in the middle iteration, – where there still exists a good amount of volume to be known but, at the same time, the large surfaces are already mostly reconstructed –, obviously decreasing from that point onward with the also decreasing of the remaining unknown volume.

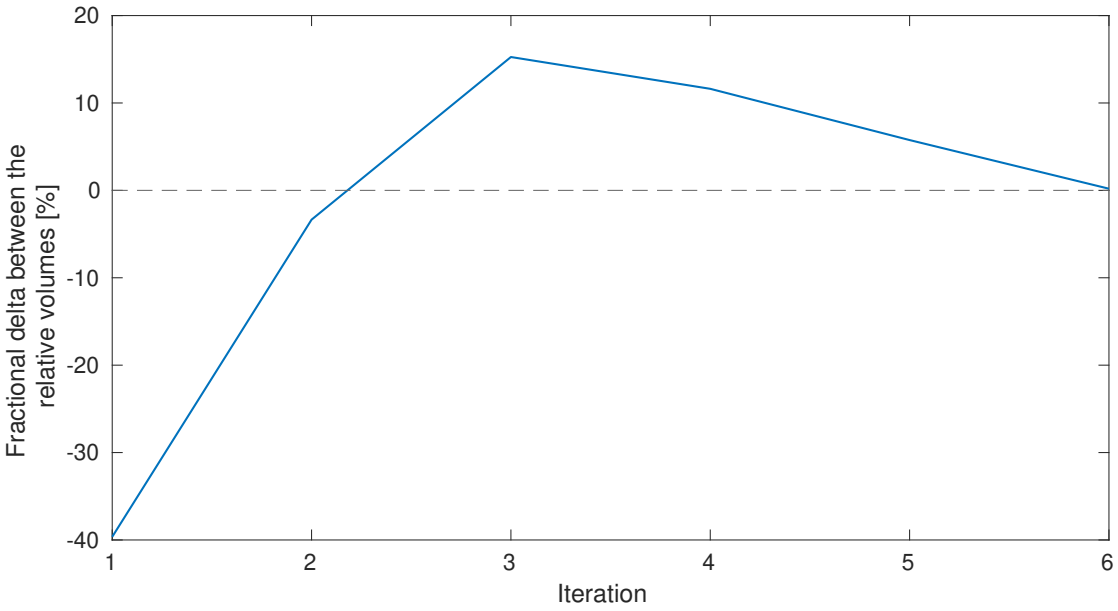


Figure 4.19: Average variation between the expected and actual fractions of unveiled volume, per iteration (Case Study 1, occluded chair scenario).

The average values only tell part of the story, since our goal is to prove that the algorithm, effectively, predicts with accuracy the least volume of a pose is capable to provide to the system. For that, Fig. 4.20 condenses all autonomous explorations data – that had information about occlusions –, correlating the volumes that are expected to be, and actually, unveiled. This means that we are only evaluating iterations that had information to predict occlusions, effectively removing the first iterations, hence ensuring that we are only comparing the robustness of our occlusion prediction algorithm. In this plot we see that a good correlation exists between the data, $R^2 = 0.9543$, seaming to prove the reliability of such predictions.

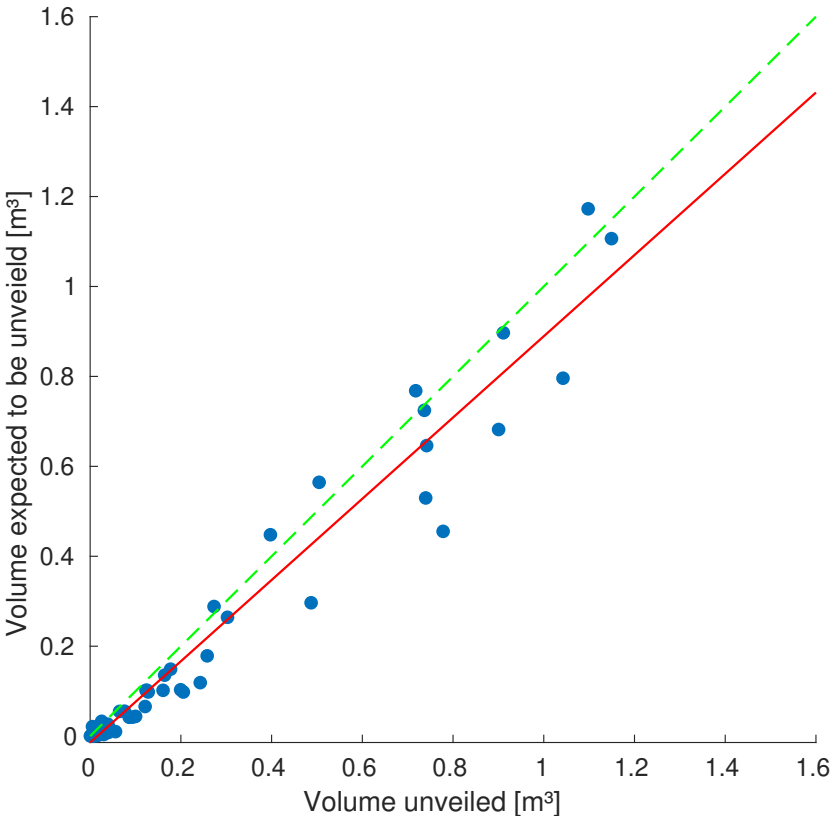


Figure 4.20: Scatter plot (for both scenarios) correlating in all iterations, except the first one, the expected and unveiled volumes. The green, dashed, line represents where the data should lay if there was a ideal correlation. The red line is the actual correlation between what is expected and what is actually measured, with $R^2 = 0.9543$.

The correlation line has a lower slope than the ideal, seaming to prove the trend of expecting to unveil a smaller volume than actually happens, due to those voxels that are evaluated while reaching for the NBV.

Case Study 1 allowed to prove the reliability of the system, capable of several autonomous explorations we are confidant that the system will, at least, reconstruct the predicted volume and will become more effective. Its high adaptability to different scenarios makes it applicable to various use cases, from an industrial bin-picking process, like the reconstruction of a newly arrived box with scattered pieces for bin-picking, to the modeling of a physical prototype, recreating it on a digital context, using our system to choose the best poses and the actual model being built with point clouds, triangle meshes, or any other desired method.

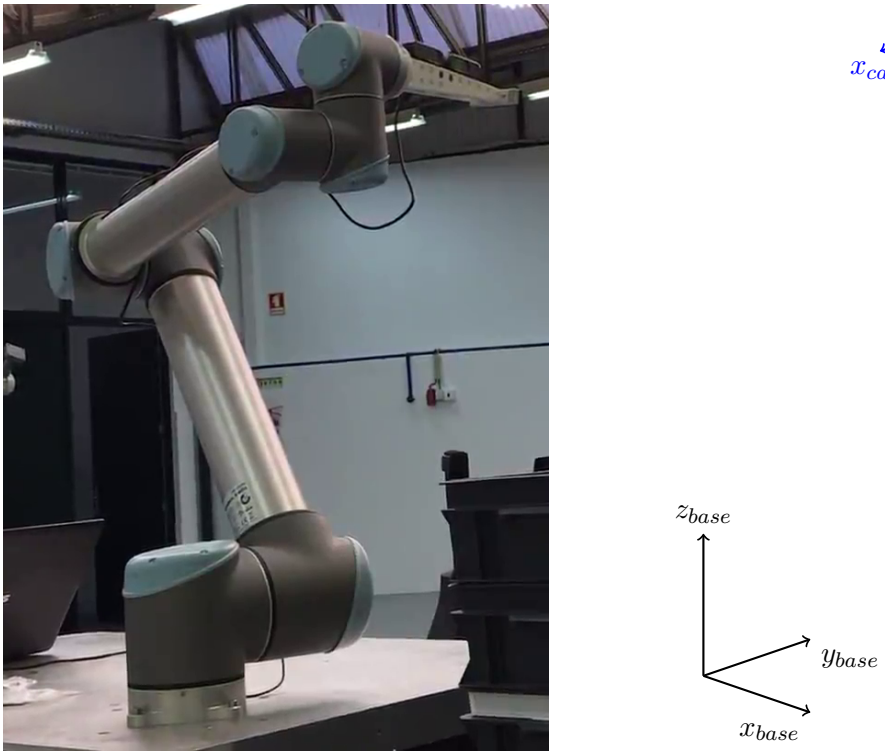
Both these examples may require different hardware, namely a different manipulator, so its important that our autonomous system maintains its characteristics in these conditions. To evaluate its performance, in a more industrial scenario, we created Case Study 2 to generate a new challenge that permits taking conclusions regarding this need.

4.2 Case Study 2: An Industrial Application

On Case Study 2 the goal was to evaluate the portability of the developed system to a different manipulator with more complex system and needs. In cooperation with Institute for Systems and Computer Engineering, Technology and Science (INESC TEC), the package was implemented on a different robotic arm – the Universal Robot's UR10 – assembled on top of a moving platform, specially designed for bin-picking tasks.

INESC TEC is a non-profit research institution, dedicated to scientific research and technological development and pre-incubation of new technology-based companies.

Comparing solely the robotic arms, the UR10 (Fig. 4.21a) has the advantage of a longer reach, up to 1.3 m, but lacks a spherical wrist, i.e., the last three rotation axis do not intersect in one single point, possibly creating some difficulties on more challenging orientations. To tackle this challenge, and following Fig. 4.21b nomenclature, an orientation restriction was implemented that consists on not allowing the X axis of the optical frame (x_{cam}) to point towards the Y axis positive end of the robot's base (y_{base}).



(a) (b)

Figure 4.21: Universal Robot's UR10 manipulator arm with the Asus Xtion PRO mounted on a special tool. In (a) is a photography of the robot, and in (b) is a scheme of the camera depth optical frame (blue axis) relative to the robot's base link (black axis).

This implementation also proved the capability of working without MoveIt, since this Application Programming Interface (API) is not used on this system. They rely on directly using the drivers which communicate with this manipulator (in this case the UR10) to send the instruction to the manipulator. By changing how poses are sampled

to, essentially, going back to the volume bounded pose sampling (section 3.12.1) and publishing the NBV to the TF tree, then communicating to the robot which frame is its goal pose, were enough changes to make it compliant with their stack. Figure 4.22 shows the scenario to explore and Fig. 4.23 is the result of this reconstruction.

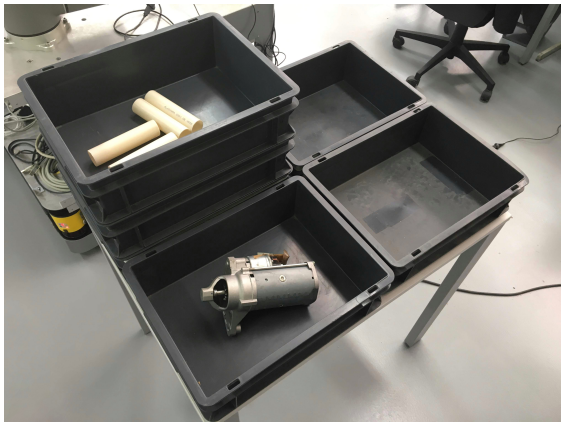


Figure 4.22: Set of boxes and their contents used for this case study. In the upper left corner box there are four plastic tubes. In the lower left box it is visible an alternator.

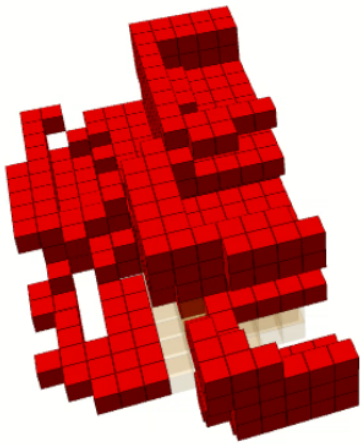


Figure 4.23: Reconstruction result from the set of boxes, with a voxel size of 50 mm.

Figure 4.24 demonstrates the process of exploration described in Case Study 2: An Industrial Application. Driven from the smaller exploration volume (1.132 m^3), compared to both scenarios in Case Study 1: Laboratory of Automation and Robotics, the reconstruction task required significantly less iterations, three in most cases. The resolution was also higher, at 50 mm, sampling fifty poses, considering that the objective was not to evaluate performance, but rather portability.

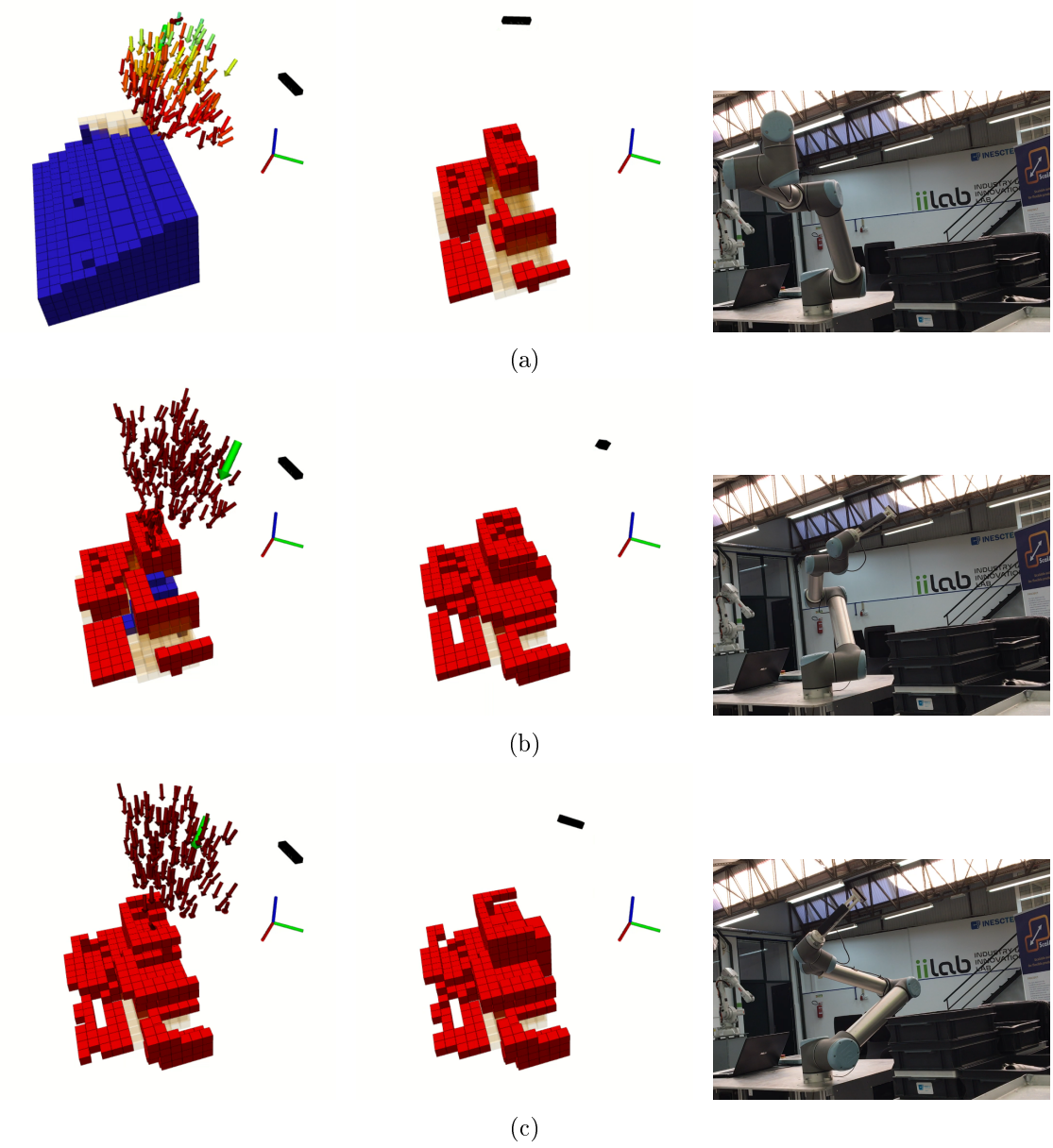


Figure 4.24: Sequence of iteration for the complete reconstruction of the scene of Case Study 2. On the left are all the possible poses (the more blue the better score, and the more red the worst), the chosen NBV (in green and bigger), the unknown voxels in orange and the voxels expected to be known on blue. In the middle is what the pose actually improves the model, red voxels are occupied, green represents the free space. On the left is the actual pose of the robot. The axis are the correspondent to the base link of the manipulator.

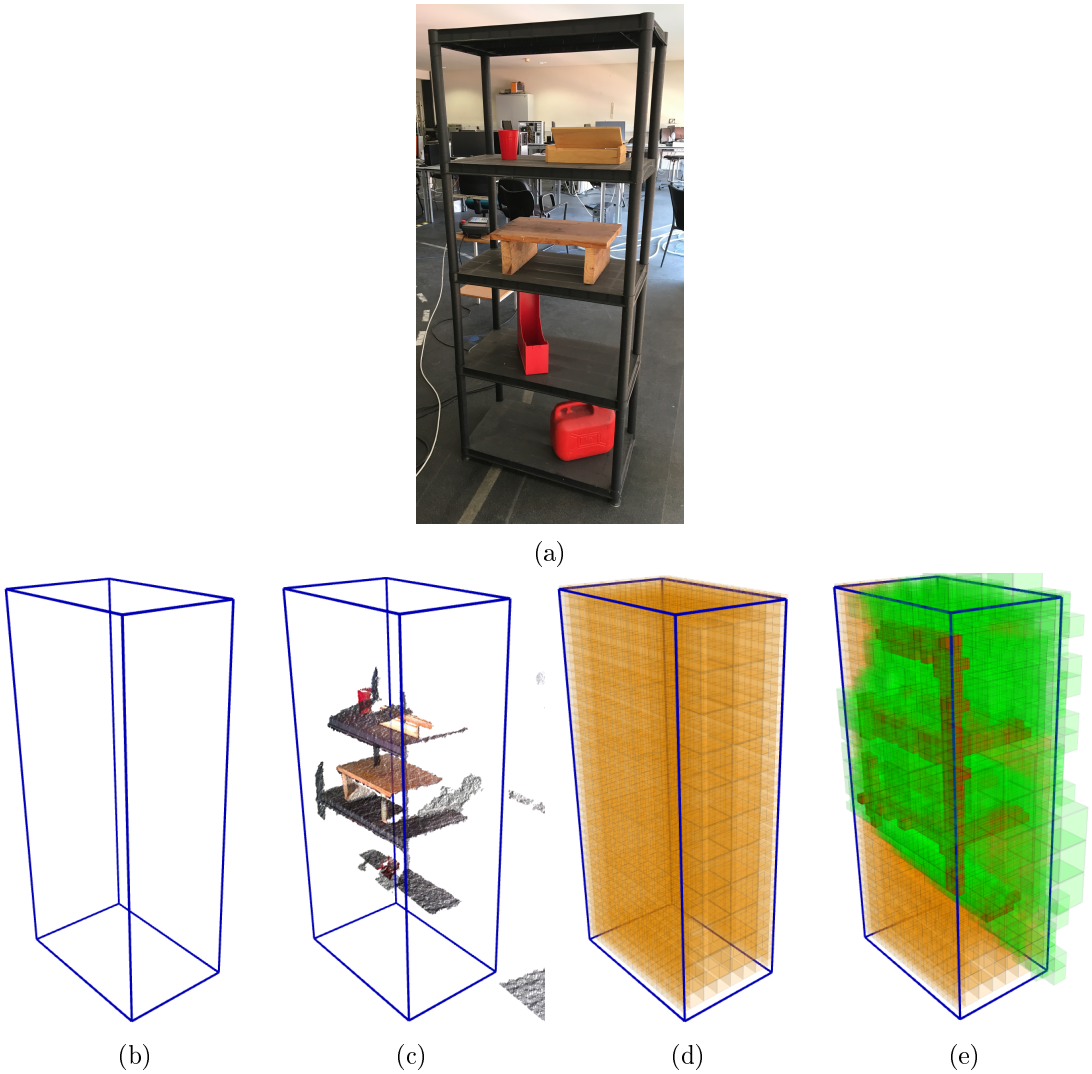
This partnership with INESC TEC proved that the developed algorithms can easily be used with other platforms, in its integrity or its constituent parts, for example, the pose sampling and evaluation algorithms. The architecture can also work in harmony with other planners, if there is the need to send pose request to the manipulator, without MoveIt.

4.3 Comparison Between Automatic and Interactive Explorations

To compare the performance of the developed algorithm against the human intelligence, a task equal to both was performed by six humans and six times by the robot.

The scenario was composed by only the shelf of the first scenario in Case Study 1 (see Fig. 4.25a). To both sets – humans and robots – the goal was to reconstruct the given scenario up to 90%. This means that the criteria to terminate the reconstruction is to achieve a reaming unknown volume less than 10% of the total exploration volume. Additionally, the users had a maximum of eight iteration – four times the best result achieved by the robot – per attempt.

The robot was allowed to use all the tools described in the previous chapters but had only one attempt. On the other hand, humans had two attempts, but the set of tools that they could use was different in each. In the first execution, the human could only see the bounding box that defined the exploration volume (Fig. 4.25b) and the colored point cloud captured by the sensor at that moment (see Fig. 4.25c), which from now on will be called the point cloud view. In the second attempt, the bounding box was still visible but instead of the point cloud, the human was allowed to observe the OcTree map being reconstructed (Figs. 4.25d and 4.25e) exactly as it is used by the robot on its procedures, known as volumetric view. The difference was that the human could only evaluate if a pose was the best one or not based on what was its visual perception.



(a) (b) (c) (d) (e)

Figure 4.25: Visualization examples of the scenario (a) and tools given to humans in order to explore the volume. The tools where the bounding box defining the exploration volume (b), the colored point cloud (c), the unknown voxels (d) and the reconstruction OcTree (e).

To make the test as fair as possible, the starting pose of the manipulator and its speed during the process were maintained constant. Because it acquires information during the movement, the interactive control was performed as described in section 3.11, and we refer to the task as completed when the 10% threshold os reached.

In average, when successfully completing the task, it takes humans a total of 6.20 iterations, using the point cloud view, lowering to 5.00 with the volumetric feedback. These values are almost double of the average robot attempt, fixed at only three iterations. Plotting the average value of volume unknown (relative to the total exploration volume) in a given iteration results on Fig. 4.26, where the autonomous system on the robot clearly minimizes – comparing to the subjects – the amount of movements requested to achieve the goal.

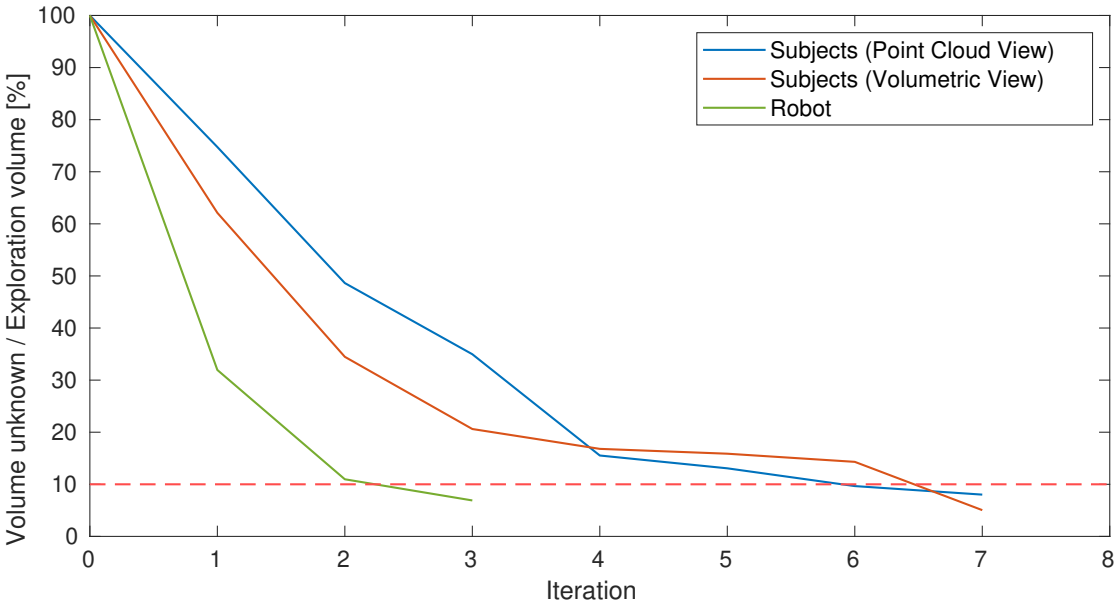


Figure 4.26: Average fraction of volume still unknown in each iteration. The red, dashed line is the cut off value.

The plot representation of Fig. 4.27 demonstrates, once more, a clear advantage of using the developed system in comparison with a user controlled robot, visualizing the point cloud data, bearing the fact that the worst robot performed better (three iterations) than the best human result (four iterations by subject 2). Furthermore, the first iteration of both robot executions revealed almost 20% more volume than the best human in his attempt (subject 4). Removing large volumes of unknown space in the first iteration is critical in order to reduce the total iterations. Given that humans did not perform well in choosing the first movement, subsequently they had the need to compensate with more iterations.

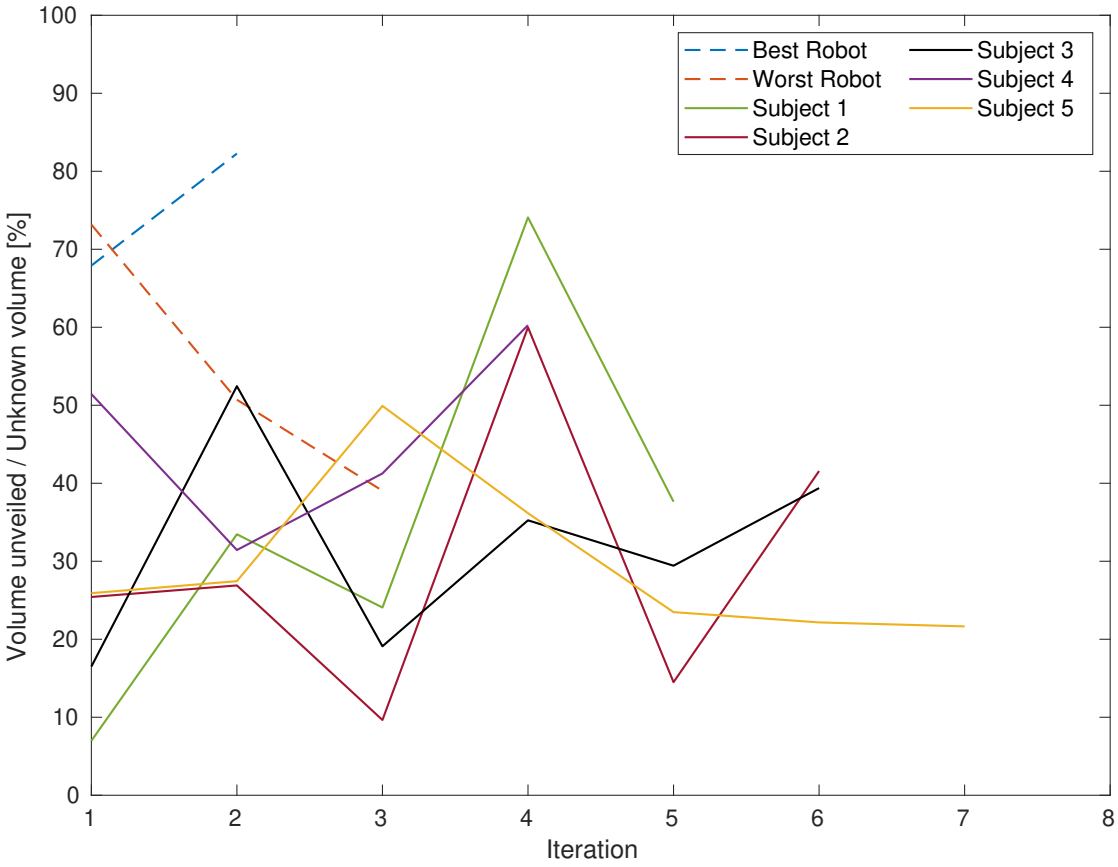


Figure 4.27: Fraction of the volume unveiled relative to the existing unknown volume, over the human and robot explorations, using only the point cloud as guidance.

In the second human attempt, when losing the point cloud but having the ability to visualize the reconstruction as it happens, both metrics improved, as on Fig. 4.28. Subject 5 was able to perform the task in the same number of movements as the worst robot, largely because their first iteration was within a 10% difference. The subject made up for this difference in the second view pose. In their final movement, the gap between the values was the biggest, at roughly 20%. The subject was actually able to beat the robot in the last iteration, regarding the fraction of unveiled volume, which makes sense observing that, in the same amount of iterations, subject 5 viewed 95.5% of the scene against 92.0% by the worst robot.

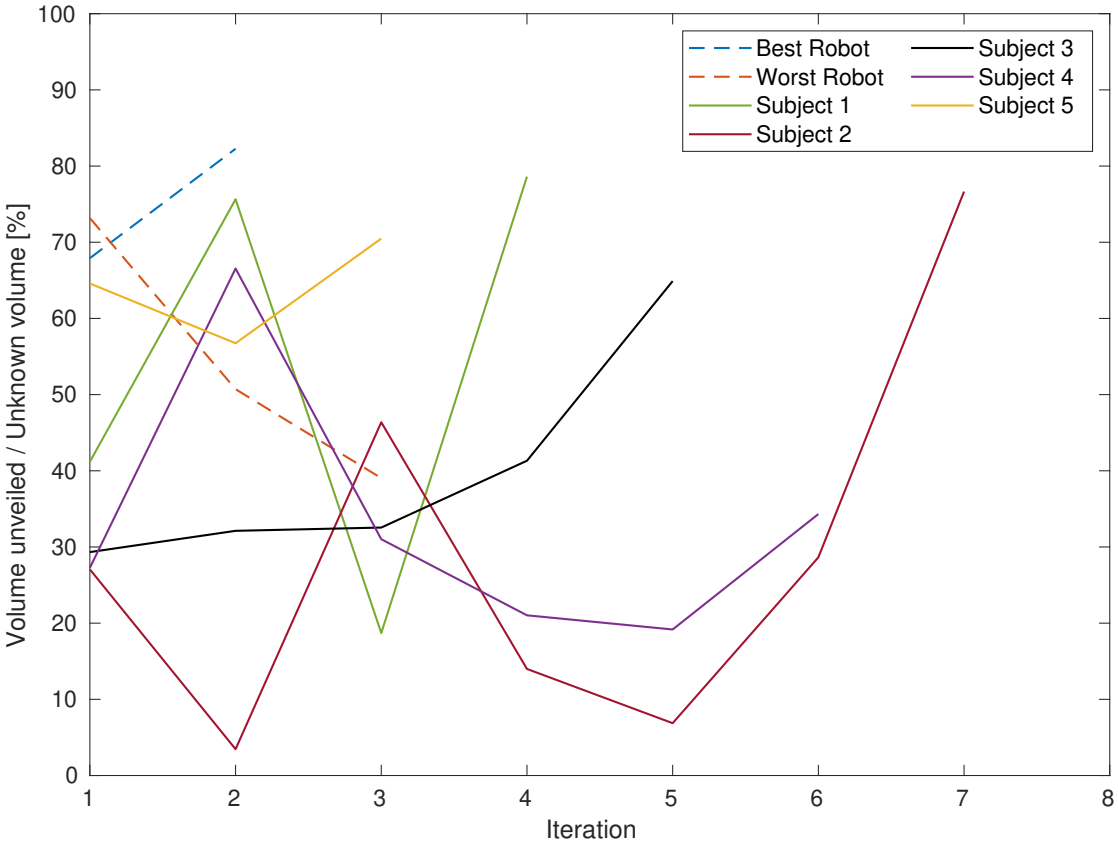


Figure 4.28: Fraction of the volume unveiled relative to the existing unknown volume, over the human and robot explorations, using the volumetric view as guidance.

In average each subject requested one less iteration for the task conclusion when having the volumetric view, which tends to indicate that even without the autonomous exploration algorithm, the tools developed help for a faster reconstruction. Further profs for this are more explicit when reviewing the attempts of the individuals that did not meet, in at least one of the executions, the requested amount of volume before the eighth iteration. Table 4.2 supports this statement, proving the evolution of individuals 6 and 7. With the volumetric set of tools, subject 6 could successfully finish the task, reconstructing more 5.7% of the shelf scene in three less movements, an improvement of 7.6% per iteration. In the other hand, subject 7 still did not complete the task, yet only changing the visualization artifact, this subject analyzed more 35.1% volume, translating to a 4.39% increase per iteration.

Table 4.2: Comparison between using the point cloud view and the volumetric view for subjects that did not achieve the goal in the maximum number of iterations. In bold are the subjects that did not achieve the goal.

Subject	Point Cloud View		Volumetric View	
	Iterations	% volume unknown	Iterations	% volume unknown
1	5	7.6	4	2.5
2	6	9.8	7	5.0
3	6	8.9	5	6.7
4	4	7.8	6	7.0
5	7	8.0	3	4.5
6	**8**	**13.6**	**5**	**7.9**
7	**8**	**50.6**	**8**	**15.5**

These prior results, regarding subjects 6 and 7, where obtained with the data plotted on Fig. 4.29. The deficit of volume gain compared to the robot process is obvious, with neither subject even matching the last iteration gains of the worst robot.

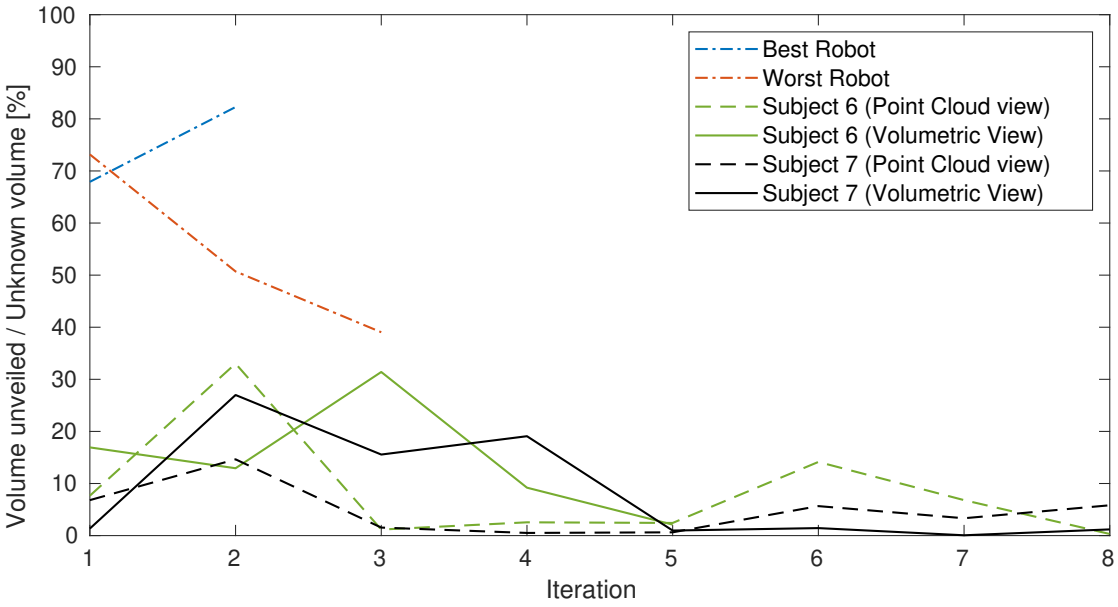


Figure 4.29: Fraction of the volume unveiled relative to the existing unknown volume, over the exploration, for subjects that did not achieve the goal in the maximum number of iterations.

Another key aspect that is interesting to compare are the tendencies of movements chosen. Two measures where taken: the euclidean distance and the rotation (based on the axis-angle representation) between two consecutive poses.

Note that the frustum section is not square, is actually a rectangle, making the roll angle of the camera influence the quantity of voxels sensed, as also does the distance to the volume. Given the fact that the exploration volume is taller than wither, a more

vertical rotation of the sensor can be preferable. Combining this effect to a given degree of yaw and/or pitch rotations can significantly increase the information gain of a pose. The said results (Table 4.3) indicate the preference by the robot to select poses that are further away and demand more from the rotational component, respectively 1.5 and 1.4 times the average human, indicating that the subjects did not account for the frustum geometry as well as the autonomous algorithm did.

Table 4.3: Comparison, per iteration, of the mean translation and rotation of the camera's depth optical frame, between the robot and human behaviors.

	Robot		Human	
	Mean	Standard Deviation	Mean	Standard Deviation
Distance [m/iteration]	1.240	0.136	0.840	0.317
Rotation [º/iteration]	127.633	22.633	91.910	28.906

The data collected during testing indicates that the autonomous system is, in fact, able to explore the volume in a more efficient manner than humans. One thing that is not measurable but occurred frequently during testing was the choosing of poses (by the algorithm) that were not intuitive at first glance, nonetheless they were quite profitable – very frequently gave the highest score of the sampled set – and made sense when gathering together all the discussed factors.

This concludes the evaluations to which the algorithms were subjected. There was seen a clear advantage of using the autonomous algorithms developed, but additional conclusions will be taken in the final chapter of this dissertation.

Intentionally blank page.

Chapter 5

Conclusions and Future Work

The main goal of this dissertation was to provide a robotic manipulator with autonomous exploration capabilities, with or without prior knowledge of the scene. For this a six Degrees of Freedom (DoF) manipulator was used with a 3D sensor rigidly connected to the end effector link.

This solution, to the best of our knowledge, is the first to try to solve the Next Best View (NBV) problem with a robotic manipulator in a real world scenario, successfully doing so.

In the process to achieve the desired solution, several algorithms had to be developed (and others improved), to match the needs of this dissertation. These algorithms can also work separately, making them applicable to other implementations. For example, having chosen OctoMap's implementation of an updatable OcTree structure, it has not explicitly stored which voxels had not their state defined yet. Tackling this issue requested the building of a parallel OcTree, generated based on the implicit information extracted from the OctoMap representation. This process is not perfect, since it always requests both structures and we have to keep in mind that the parallel OcTree has voxels marked as free that are corresponding to unknown in the original one. Yet, considering the lack of tools for this procedure, the algorithm worked as pretended given the fact that we were able to successfully – and reliably – use this last OcTree to known which voxels needed to be visited.

Knowing what needs to be reveled, by itself, does not contribute to the solving of the NBV problem. The solution implemented in this work integrates this with the sampling of several poses, which is performed considering the robot characteristics as well as where the volumes are most aggregated, in an effort to make the poses plausible. The idea behind generating only plausible poses is to, effectively, evaluate poses that are guaranteed to be looking towards some amount of unknown space. With that said, not all poses will observe the same amount of unknown voxels. To choose the one capable of perceiving the largest amount possible of the environment, a procedure capable of estimating which voxels would potentially be intercepted by the depth sensor rays and, hence, how much volume would be unveiled, is executed for each one. This procedure gives us a metric to rank the poses. To successfully access how much volume is expected to be unveiled, the portions of it that are occluded have to be predicted and not accounted for. This means that, if there is an occupied volume between the camera and a voxel in an unknown state, this voxel must not be accounted as possibly visible, as this is actually not possible, from the given pose.

The algorithm proved to be reliable and stable, in the sense that for the same scene, the solutions where different due to the randomness of the pose sampling, but the amount of iterations never diverged significantly. By autonomously exploring, the system is able to perceive when it stands before volumes that it can not reach for evaluating, moving away from them as they do not provide novel information.

In addition, we also compared the exploration performance of the robot to that of human users. The robot was able to reconstruct a scenario in half the iterations required, in average, by the set of humans.

The developed and improved ROS packages regarding this dissertation are publicly available:

- SmObEx - Smart Object Exploration: `https://github.com/lardemua/SmObEx`
- FANUC M6iB/6S Support and MoveIt Config: `https://github.com/ros-industrial/fanuc/pull/264`
- OctoMap tools: `https://github.com/miguelriemoliveira/octomap_tools`

Also, a list of videos of the working system are accessible:

- Hand-eye extrinsic calibration: `https://youtu.be/zZ-sPsrrcI0`
- Mapping of only a selected portion of the world: `https://youtu.be/paOhtI7LZPg`
- Interactive volume reconstruction: `https://youtu.be/-pPXkNzlAXI`
- Interactive pose evaluation and scene reconstruction: `https://youtu.be/ltMPFWkhAAE`
- Fully autonomous system exploring both case scenarios: `https://youtu.be/gfenHzBJkGk`
- Video resume of both interactive and autonomous modes: `https://youtu.be/pqGjP4bn5YY`

Even though the main goal of this dissertation, along with the necessary milestones, were effectively and satisfactorily reached, some improvement can yet be executed in addition to more industrial performance adaption and testing.

We decided not to use a cost component in the score formula (eq. 3.2) given that, in energetic terms, did not make sense considering that our proposed goal was the development of a exploration system efficient in the number of poses requested, rather than an energetically efficient one. Yet, some implementations should be done to avoid the movement of the manipulator to extreme poses, giving preference to a more organized and intuitive path. Building upon this could be the use of an optimization heuristic – like a genetic algorithm – that returned the set of poses that should be visited in a sequence that, as wanted, would be more structured. This would eliminate the possible need of evaluating a complete path, since the optimization should predict the most profitable one.

This dissertation also creates the foundations to several other works. The NBV selecting algorithm could be used in a more industrial oriented application, as is bin-picking. In these kind of tasks, removing one piece can help unveil others, that then become more easily accessible. Using the developed system for pose selection, the manipulator could

force the passage in a waypoint that would maximize its knowledge about the constantly changing scene.

One other branch from this work could be a vehicle capable of exploring any ambient, by mounting the manipulator in a moving platform. With a path planner developed to integrate both platform and manipulator, the pose generation would not be bounded by the manipulator's reach by but instead only by the volume requested for the exploration. The platform plus manipulator would be able to move to the selected NBV, avoiding obstacles in the way.

Lastly, possible combining the RGB data, the robot could detect a new object placed before it, to then classify it. This classification would be useful for different tasks, for example, choosing the proper gripper configuration to grab it or to place it in the desired container, if we think of a task where the goal of the robot is to separate the objects by categories.

Regarding testing, we proved that the developed algorithm is capable of unveiling more volume than it predicted for. Notwithstanding, we gathered no information concerning the actual voxels expected to be known and the ones actually measured. Excepting a visual examination, we gathered no quantitative data that guarantees that the voxels observed are exactly the ones predicted, since various, distinct, voxels can accumulate to the same amount of volume. In further testing, this should contribute to the full understanding of the prediction accuracy.

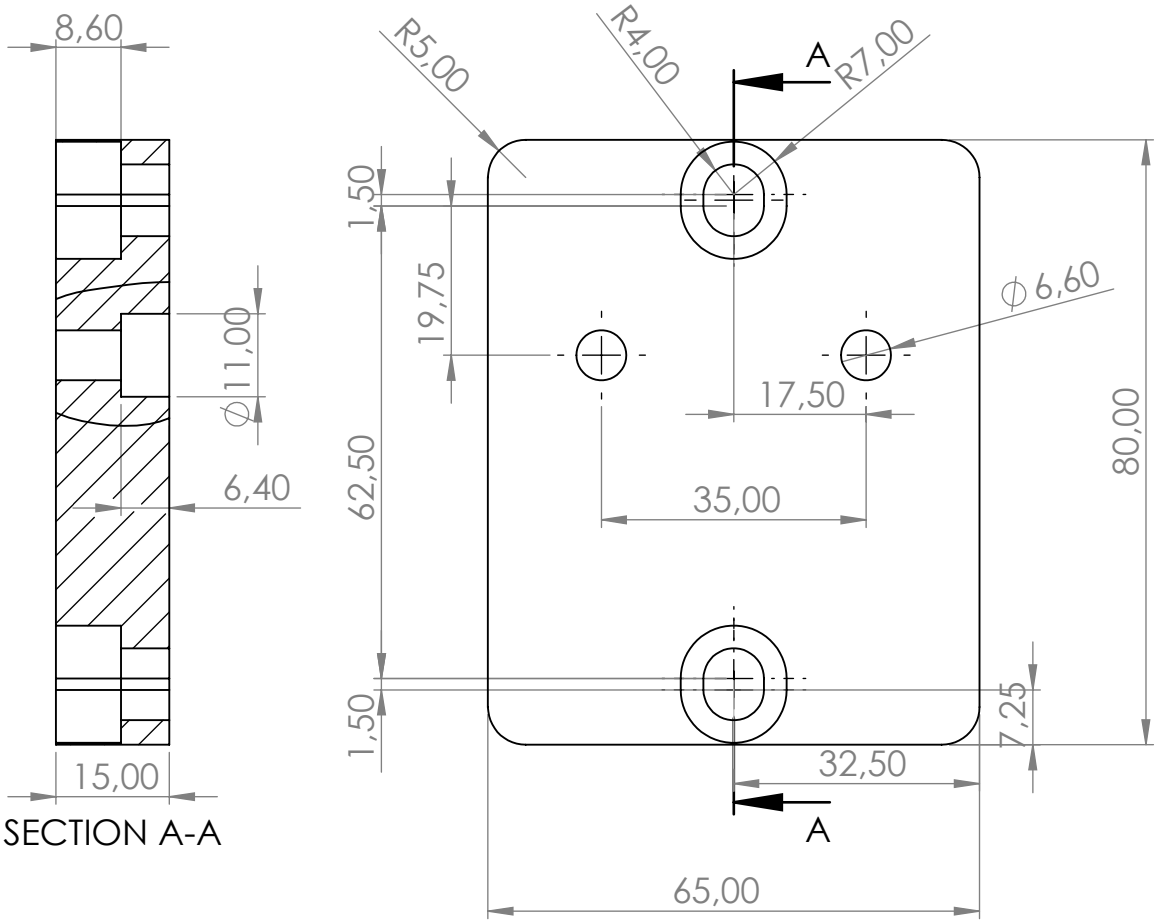
To make it even more robust, the system could detect changes in the scene it already knows (for example, placing or removing an object). This detection should make possible for the surroundings of that volume to be remarked as unknown, requesting once again its evaluation, which in turn would improve the adaptability and integrity of the model.

Bearing everything in mind, the contributions done by this work have a large potential for several industrial, but also for prototype development, being considered by the author as a great starting point in the path to provide manipulators with awareness about their surroundings.

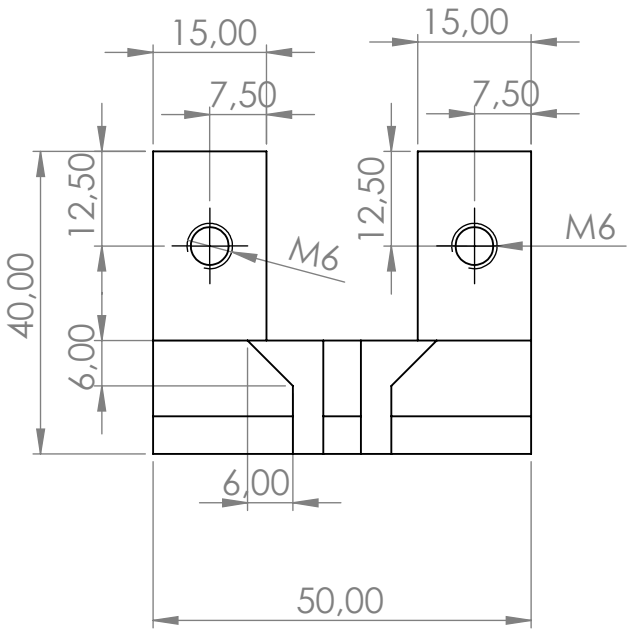
Intentionally blank page.

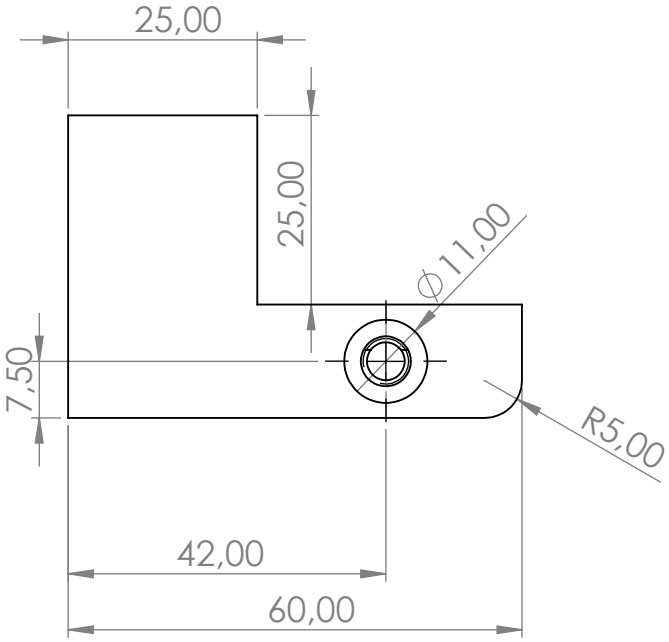
Appendix A

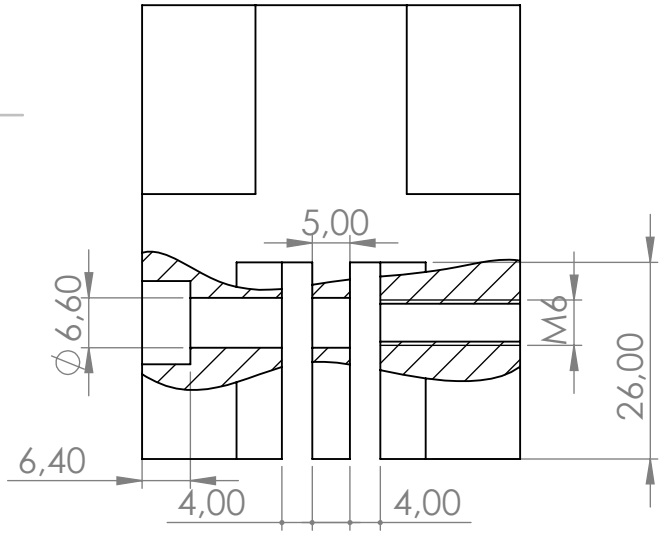
Technical Drawings

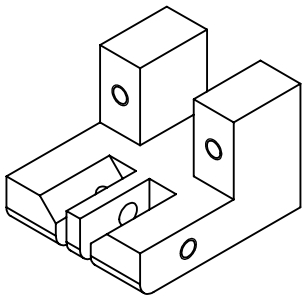


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH: ISO 2768-m		DEBURR AND BREAK SHARP EDGES			DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
						BaseF	
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1









UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH: ISO 2768-m	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE:

MATERIAL:

DWG NO. SuporteF

A4

WEIGHT:

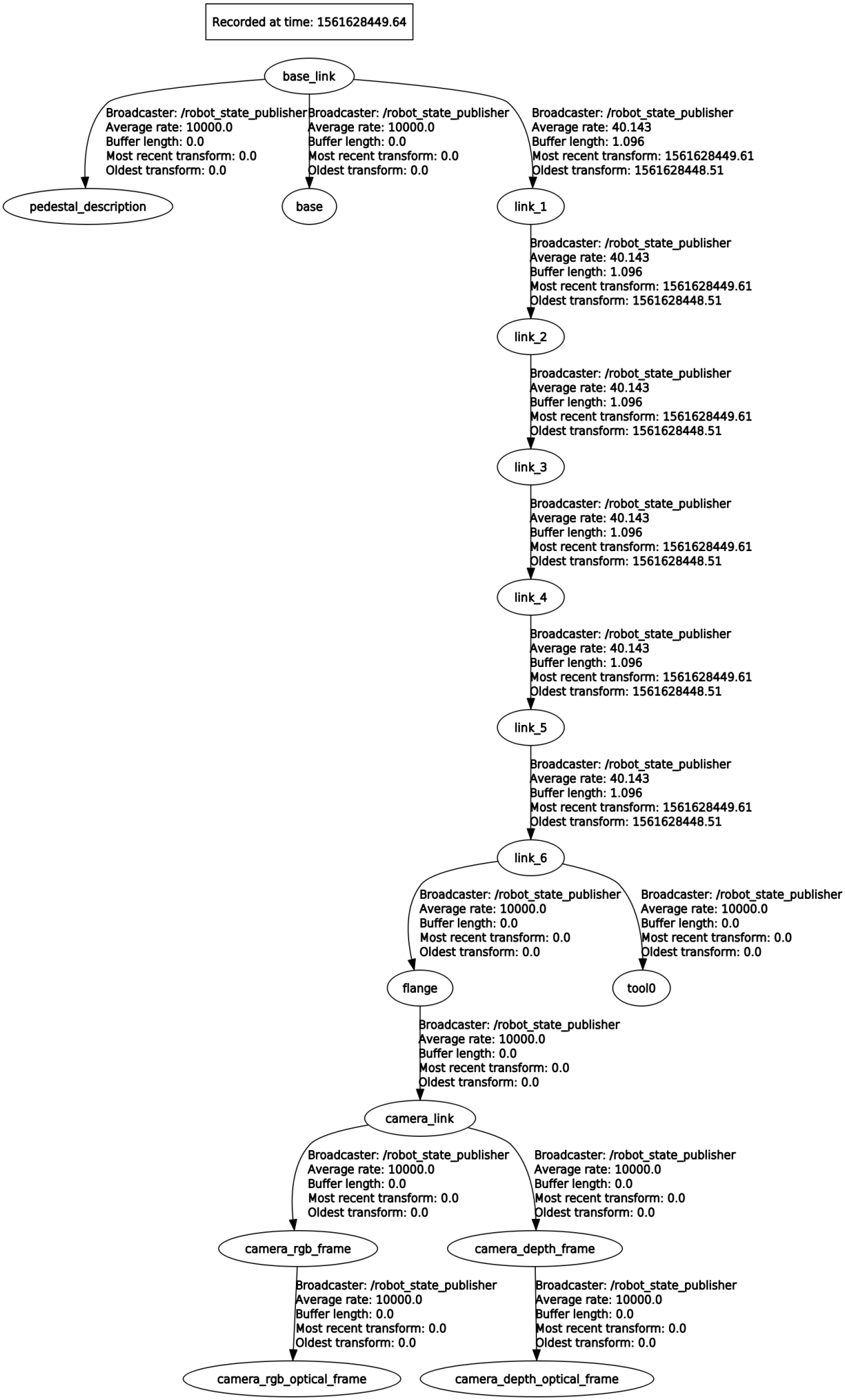
SCALE:1:1

SHEET 1 OF 1

Intentionally blank page.

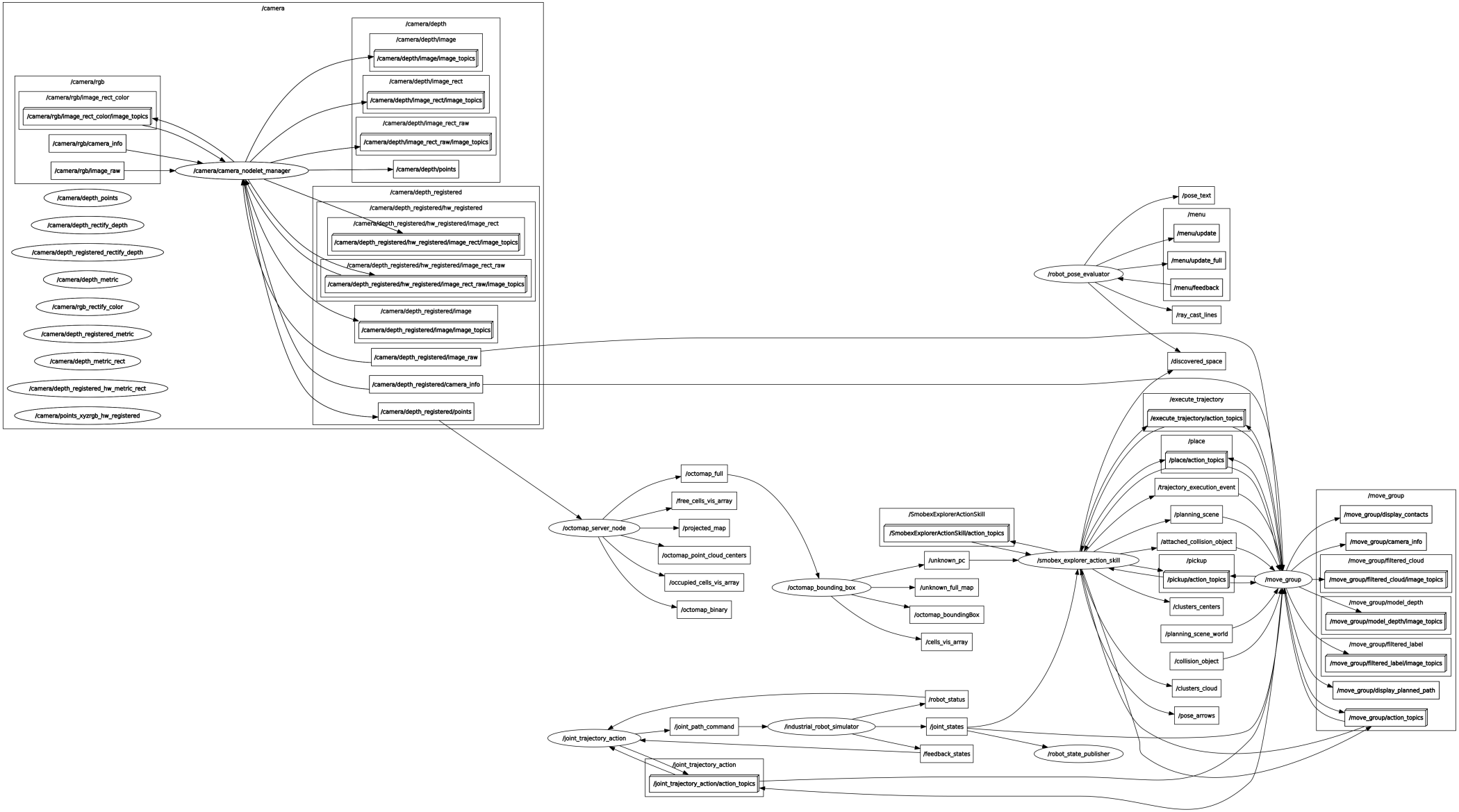
Appendix B

Transformations Tree


Recorded at time: 1561628449.64
base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
pedestal_description
base
link_1
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
link_2
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
link_3
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
link_4
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
link_5
Broadcaster: /robot_state_publisher
Average rate: 40.143
Buffer length: 1.096
Most recent transform: 1561628449.61
Oldest transform: 1561628448.51
link_6
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
flange
tool0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_rgb_frame
camera_depth_frame
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_rgb_optical_frame
camera_depth_optical_frame

Appendix C

Node Graph


/camera
/camera/depth
/camera/depth/image
/camera/depth/image/image_topics
/camera/depth/image_rect
/camera/depth/image_rect/image_topics
/camera/depth/image_rect_raw
/camera/depth/image_rect_raw/image_topics
/camera/depth/points
/camera/rgb
/camera/rgb/image_rect_color
/camera/rgb/image_rect_color/image_topics
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/camera_nodelet_manager
/camera/depth_registered
/camera/depth_registered/hw_registered
/camera/depth_registered/hw_registered/image_rect
/camera/depth_registered/hw_registered/image_rect/image_topics
/camera/depth_registered/hw_registered/image_rect_raw
/camera/depth_registered/hw_registered/image_rect_raw/image_topics
/camera/depth_registered/image
/camera/depth_registered/image/image_topics
/camera/depth_registered/image_raw
/camera/depth_registered/camera_info
/camera/depth_registered/points
/camera/depth_points
/camera/depth_rectify_depth
/camera/depth_registered_rectify_depth
/camera/depth_metric
/camera/rgb_rectify_color
/camera/depth_registered_metric
/camera/depth_metric_rect
/camera/depth_registered_hw_metric_rect
/camera/points_xyzrgb_hw_registered
/pose_text
/menu
/menu/update
/menu/update_full
/menu/feedback
/ray_cast_lines
/robot_pose_evaluator
/discovered_space
/execute_trajectory
/execute_trajectory/action_topics
/place
/place/action_topics
/trajectory_execution_event
/planning_scene
/attached_collision_object
/pickup
/pickup/action_topics
/clusters_centers
/planning_scene_world
/collision_object
/clusters_cloud
/pose_arrows
/move_group
/move_group/display_contacts
/move_group/camera_info
/move_group/filtered_cloud
/move_group/filtered_cloud/image_topics
/move_group/model_depth
/move_group/model_depth/image_topics
/move_group/filtered_label
/move_group/filtered_label/image_topics
/move_group/display_planned_path
/move_group/action_topics
/move_group
/octomap_full
/free_cells_vis_array
/projected_map
/octomap_point_cloud_centers
/occupied_cells_vis_array
/octomap_binary
/octomap_server_node
/SmobexExplorerActionSkill
/SmobexExplorerActionSkill/action_topics
/unknown_pc
/smobex_explorer_action_skill
/octomap_bounding_box
/unknown_full_map
/octomap_boundingBox
/cells_vis_array
/robot_status
/joint_path_command
/industrial_robot_simulator
/joint_states
/joint_trajectory_action
/feedback_states
/robot_state_publisher
/joint_trajectory_action
/joint_trajectory_action/action_topics

Bibliography

[Arrais *et al.* 2017] Rafael Arrais, Miguel Oliveira, César Toscano and Germano Veiga. A mobile robot based sensing approach for assessing spatial inconsistencies of a logistic system. *Journal of Manufacturing Systems*, 43:129–138, April 2017.

[Blodow *et al.* 2011] Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, Thomas Ruhr, Moritz Tenorth and Michael Beetz. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4263–4270, San Francisco, CA, September 2011. IEEE.

[Brito Junior *et al.* 2002] A.M. Brito Junior, L.M. Goncalves, G. Tho and A.L. De O Cavalcanti. A simple sketch for 3D scanning based on a rotating platform and a Web camera. In *Proceedings. XV Brazilian Symposium on Computer Graphics and Image Processing*, p. 406, Fortaleza-CE, Brazil, 2002. IEEE Comput. Soc.

[Chitta *et al.* 2012] Sachin Chitta, Ioan Sucan and Steve Cousins. MoveIt! [ROS Topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, March 2012.

[Condurache and Burlacu 2016] D. Condurache and A. Burlacu. Orthogonal dual tensor method for solving the A X = X B sensor calibration problem. *Mechanism and Machine Theory*, 104:382–404, October 2016.

[Dornhege and Kleiner 2011] Christian Dornhege and Alexander Kleiner. A frontier-void-based approach for autonomous exploration in 3d. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 351–356, Kyoto, Japan, November 2011. IEEE.

[Douillard *et al.* 2010] B Douillard, J Underwood, N Melkumyan, S Singh, S Vasudevan, C Brunner and A Quadros. Hybrid elevation maps: 3D surface models for segmentation. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1532–1538, Taipei, October 2010. IEEE.

[Evers and Naylor 2018] Christine Evers and Patrick A. Naylor. Acoustic SLAM. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1484–1498, September 2018.

[FANUC 2007] FANUC. M-6iB Series, 2007.

[Gedicke *et al.* 2016] Thorsten Gedicke, Martin Günther and Joachim Hertzberg. FLAP for CAOS: Forward-Looking Active Perception for Clutter-Aware Object Search. p. 6, 2016.

[Heller *et al.* 2014] Jan Heller, Didier Henrion and Tomas Pajdla. Hand-eye and robot-world calibration by global polynomial optimization. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3157–3164, Hong Kong, China, May 2014. IEEE.

[Hepp *et al.* 2018] Benjamin Hepp, Debadeepta Dey, Sudipta N. Sinha, Ashish Kapoor, Neel Joshi and Otmar Hilliges. Learn-to-Score: Efficient 3D Scene Exploration by Predicting View Utility. in Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu and Yair Weiss, editores, *Computer Vision – ECCV 2018*, Vol. 11219, pp. 455–472. Springer International Publishing, Cham, 2018.

[Hornung *et al.* 2013] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss and Wolfram Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, April 2013.

[Hull 2017] Graham Hull. Real-time occupancy grid mapping using LSD-SLAM. Stellenbosch university, December 2017.

[Isler *et al.* 2016] Stefan Isler, Reza Sabzevari, Jeffrey Delmerico and Davide Scaramuzza. An information gain formulation for active volumetric 3D reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3477–3484, Stockholm, Sweden, May 2016. IEEE.

[Kriegel 2015] Simon Kriegel. Autonomous 3D Modeling of Unknown Objects for Active Scene Exploration. 2015.

[Kriegel *et al.* 2013] Simon Kriegel, Manuel Brucker, Zoltan-Csaba Marton, Tim Bodenmuller and Michael Suppa. Combining object modeling and recognition for active scene exploration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2384–2391, Tokyo, November 2013. IEEE.

[Kuffner and LaValle 2000] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, Vol. 2, pp. 995–1001, San Francisco, CA, USA, 2000. IEEE.

[Lavalle 1998] Steven M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

[Li *et al.* 2010] Aiguo Li, Lin Wang and Defeng Wu. Simultaneous robot-world and hand-eye calibration using dual-quaternions and Kronecker product. p. 8, September 2010.

[Maegher 1980] Donald Maegher. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, October 1980.

[Park and Martin 1994] Frank Chongwoo Park and Bryan J. Martin. Robot sensor calibration: solving AX=XB on the Euclidean group. *IEEE Transactions on Robotics and Automation*, 10:717–721, October 1994.

[Pfaff and Burgard 2006] Patrick Pfaff and Wolfram Burgard. An Efficient Extension of Elevation Maps for Outdoor Terrain Mapping. In *Field and Service Robotics*, Vol. 25, pp. 195–206. Springer Berlin Heidelberg, 2006.

[Rusu and Cousins 2011] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pp. 1–4, Shanghai, China, May 2011. IEEE.

[Sarmiento *et al.* 2003] A. Sarmiento, R. Murrieta and S.A. Hutchinson. An efficient strategy for rapidly finding an object in a polygonal world. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Vol. 2, pp. 1153–1158, Las Vegas, NV, USA, 2003. IEEE.

[Shah 2013] Mili Shah. Solving the Robot-World/Hand-Eye Calibration Problem Using the Kronecker Product. *Journal of Mechanisms and Robotics*, 5(3):031007, June 2013.

[Shiu and Ahmad 1989] Y. C. Shiu and Shaheen Ahmad. Calibration of Wrist-Mounted Robotic Sensors by Solving Homogeneous Transform Equations of the Form AX=XB. *IEEE Transactions on Robotics and Automation*, 5:16–29, March 1989.

[Stoyanov *et al.* 2010] Todor Stoyanov, Martin Magnusson, Henrik Andreasson and Achim J Lilienthal. Path planning in 3D environments using the Normal Distributions Transform. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3263–3268, Taipei, October 2010. IEEE.

[Strobl and Hirzinger 2006] Klaus Strobl and Gerd Hirzinger. Optimal Hand-Eye Calibration. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4647–4653, Beijing, China, October 2006. IEEE.

[Tabb and Ahmad Yousef 2017] Amy Tabb and Khalil M. Ahmad Yousef. Solving the robot-world hand-eye(s) calibration problem with iterative methods. *Machine Vision and Applications*, 28(5-6):569–590, August 2017.

[Tan *et al.* 2018] Ning Tan, Xiaoyi Gu and Hongliang Ren. Simultaneous Robot-World, Sensor-Tip, and Kinematics Calibration of an Underactuated Robotic Hand With Soft Fingers. *IEEE Access*, 6:22705–22715, 2018.

[Vanneste *et al.* 2014] Simon Vanneste, Ben Bellekens and Maarten Weyn. 3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap. p. 13, 2014.

[Vasquez-Gomez and Romero 2019] Juan Irving Vasquez-Gomez and David Ernesto Troncoso Romero. Next-Best-View Regression using a 3D Convolutional Neural Network. p. 6, 2019.

[Yervilla-Herrera *et al.* 2018] Heikel Yervilla-Herrera, J. Irving Vasquez-Gomez, Rafael Murrieta-Cid, Israel Becerra and L. Enrique Sucar. Optimal motion planning and stopping test for 3-D object reconstruction. *Intelligent Service Robotics*, 12(1):103–123, 2018.